Der Dominoeffekt in der Fehlerbehandlung von Prozeßsystemen

von

Peter Bathelt

Erlangen - 1986

1987 № 12

UNIV.
LEIDEN
BIBL.

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41538126_0003

Der Dominoeffekt in der Fehlerbehandlung von Prozeßsystemen

Der Technischen Fakultät der
Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Peter Bathelt

Erlangen - 1986

Als Dissertation genehmigt von der
Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	30.04.1986
Tag der Promotion:	01.09.1986
Dekan:	Prof. Dr. H. Brand
Berichterstatter:	Prof. Dr. F. Hofmann
	Prof. Dr. H. Wedekind

Abstract

Bathelt, Peter:

Der Dominoeffekt in der Fehlerbehandlung von Prozeßsystemen

In Systemen nebenläufiger Prozesse, die gegenseitigen Wechselwirkungen unterliegen, besteht im Fehlerfall, bei Verwendung von Rücksetzstrategien zur Fehlerkorrektur, die Gefahr des Dominoeffektes, d.h. die Gefahr der Rücksetzung einer nicht vorab begrenzbaren Anzahl von Aktionen der aktuellen Berechnungsfolge, die zum fehlerhaften Prozeßsystem-Zustand geführt hat.

In der vorliegenden Arbeit werden, unter Zugrundelegung eines formalen Prozeßsystem-Modells und eines entsprechenden Fehlerbehandlungs-Modells (Recovery-Modells), hinreichende und notwendige Bedingungen für das Auftreten des Dominoeffektes abgeleitet.
Je nach dem, ob im Recovery-Fall potentielle Fehlerverursacher berücksichtigt und zurückgesetzt werden oder nicht, wird gezeigt, daß der Dominoeffekt insbesondere durch eine nicht ausreichend auf die gegenseitigen Wechselwirkungen nebenläufiger Prozesse abgestimmte Strategie zum Anlegen prozeßlokaler Wiederaufsetzpunkte bzw., unter der Voraussetzung der Existenz adäquater Akzeptanztests, durch eine ungenügend koordinierte Vorgehensweise bei der Verifikation aktueller Prozeßsystem-Zustände begünstigt wird.

Die Ableitung dieser Ergebnisse ermöglicht die Identifizierung verschiedener dominoeffektfreier Prozeßsystem-Klassen und die Herleitung alternativer Verfahrensweisen zur Konstruktion dominoeffektfreier Systeme. Insbesondere wird in diesem Zusammenhang vorgeschlagen, in Analogie zum Deadlock-Problem Strategien zur Dominoeffekt-Vermeidung, -Verhinderung und -Erkennung mit anschließender Beseitigung abzuleiten.

Das Beschreiben hinreichender und notwendiger Bedingungen für das Auftreten des Dominoeffektes wird erst durch die Charakterisierung der Menge aller im Fehlerfall zurückzusetzenden Aktionen einer Berechnungsfolge ermöglicht. Diese Präzisierung beruht auf einer exakten Definition der Begriffe "berechnungstechnisch/syntaktisch konsistenter" und "fehlerfreier" Prozeßsystem-Zustand, dessen Formalisierung die Einführung und Präzisierung des Begriffes einer "Recoveryfunktion" ermöglicht.
Mit "Recoveryfunktion" wird eine Vorschrift bezeichnet, mit deren Hilfe ausgehend von einem fehlerhaften Prozeßsystem-Zustand ein berechnungstechnisch/syntaktisch konsistenter und in einem noch näher zu beschreibenden Sinne fehlerfreier Prozeßsystem-Zustand wiederhergestellt werden kann. Ihre Beschreibung stellt einen wesentlichen Beitrag dieser Arbeit dar, denn unter ihrer Verwendung können Ergebnisse zur Dominofreiheit von Prozeßsystemen insbesondere auch unter Berücksichtigung der zugrundeliegenden Fehlerbehandlungs-Strategien formal bewiesen werden.

Mein besonderer Dank gilt Herrn Prof. Dr. F. Hofmann, der diese Arbeit ermöglicht hat. Durch zahlreiche wertvolle Hinweise und Anregungen hat er maßgeblich zu ihrem Gelingen beigetragen.

Für die Übernahme des Zweitgutachtens danke ich Herrn Prof. Dr. H. Wedekind.

Herrn Dr. H. Schulze von der Firma Siemens, Abteilung E STE 36, danke ich für die Erlaubnis, zur Anfertigung der Arbeit die firmeneigenen Textverarbeitungssysteme nutzen zu dürfen.

INHALTSVERZEICHNIS

1. Einleitung und Übersicht 1

1.1 Motivation 1
1.2 Abgrenzung des Themas 4
1.3 Einleitende Beispiele 7
1.4 Wesentliche Ergebnisse 19
1.5 Übersicht zum Aufbau 20

2. Zugrundeliegendes Modell 21

2.1 Prozeßsystem-Modell 21

2.1.1 Formalisierung wesentlicher Begriffe *21*

2.1.1.1 Grundlegende Vereinbarungen *21*
2.1.1.2 Prozeßsystem-Begriff *23*
2.1.1.3 Berechnungs-Begriff *24*
2.1.1.4 Veranschaulichung der Begriffe *27*

2.1.2 Aktionentypen *32*

2.1.2.1 Lesende und schreibende Aktionen *32*
2.1.2.2 Lokale und globale Aktionen *34*
2.1.2.3 Nachrichtenübertragende Aktionen *37*
2.1.2.4 Synchronisierende Aktionen *41*

2.1.3 Aktions- und Prozeßabhängigkeiten *47*

2.1.3.1 Basiert auf Relation *47*
2.1.3.2 Lokale und globale Abhängigkeiten *48*
2.1.3.3 Kommunikationsbedingte Abhängigkeiten *50*
2.1.3.4 Koordinierungsbedingte Abhängigkeiten *50*

2.1.4 Prozeßsystemklassen *52*

2.1.4.1 Systeme unabhängiger Prozesse *52*
2.1.4.2 Systeme kooperierender Prozesse *53*
2.1.4.3 Systeme kommunizierender Prozesse *54*
2.1.4.4 Systeme koordinierender Prozesse *55*

2.2 Fehlerbehandlungs-Modell 57

2.2.1 Unterlagertes Rechensystem 57

2.2.1.1 Systemkomponenten 57
2.2.1.2 Prozeßsystemeinbettung 64
2.2.1.3 Funktionsweise des Rechensystems 65
2.2.1.4 Verhalten im Fehlerfall 66

2.2.2 Vorausgesetztes Zustandssicherungskonzept 69

2.2.2.1 Historie prozeßlokaler Wiederaufsetzpunkte 69
2.2.2.2 Historie globaler Variablenwerte 71
2.2.2.3 Zugrundegelegte Speicherungsstrukturen 72
2.2.2.4 Veranschaulichung des Zustandssicherungskonzeptes 73

2.2.3 Zugrundegelegtes Wiederaufsetzverfahren 85

2.2.3.1 Zu beachtende Fehlerfälle 85
2.2.3.2 Allgemeine Wiederanlaufprozedur 87
2.2.3.3 Erläuterung am Beispiel 94
2.2.3.4 Petri-Netz Darstellung eines Fehlerbehandlungsvorgangs 99

2.2.4 Berücksichtigung potentieller Fehlerverursacher 111

2.2.4.1 Zu beachtende Fehlerfälle 111
2.2.4.2 Allgemeine Wiederanlaufprozedur 115
2.2.4.3 Erläuterung am Beispiel 117
2.2.4.4 Petri-Netz Darstellung eines Fehlerbehandlungsvorgangs 123

3. Recoveryfunktionskonzept 134

3.1 Abstraktion vom Fehlerbehandlungskonzept 134

3.1.1 Endezeitpunkt einer Aktion 134
3.1.2 Aktuellster Wiederaufsetzpunkt 136
3.1.3 Begriff der Recoveryeinheit 137
3.1.4 Einheitsrelation 138

3.2 Beschreibung der Menge zurückzusetzender Aktionen 140

3.2.1 Fehlerhafte und ungesicherte Aktionen 140
3.2.2 Potentielle Fehlerverursacher 141
3.2.3 Verifizierte Aktionen 142
3.2.4 Zurückzusetzende Aktionen 144

3.3 Klassifizierung von Prozeßsystem-Zuständen 151

3.3.1 Konsistente Prozeßsystem-Zustände 151
3.3.2 Fehlerhafte und ungesicherte Prozeßsystem-Zustände 151
3.3.3 Verifizierte Prozeßsystem-Zustände 152
3.3.4 Korrektur fehlerhafter Prozeßsystem-Zustände 153

3.4 Ableitung von Recoveryfunktionen 155

3.4.1 Anforderungen an Recoveryfunktionen 155
3.4.2 Allgemeine Recoveryfunktion 157
3.4.3 Koordinierungsbezogene Recoveryfunktion 163
3.4.4 Kommunikationsbezogene Recoveryfunktion 171

4. Charakterisierung des Dominoeffektes 176

4.1 Unterscheidung zweier Teilprobleme 176

4.2 Dominoeffekt und Rücksetzausbreitung 181

4.2.1 Unvermeidbare und vermeidbare Rücksetzausbreitung 181
4.2.2 Betrachtungen zur Begrenzung der Rücksetzausbreitung 183
4.2.3 Begrenzung der vermeidbaren Rücksetzausbreitung 193
4.2.4 Begrenzung der Zahl zurückzusetzender Aktionen 202

4.3 Dominoeffekt und Fehlerausbreitung 205

4.3.1 Vorüberlegungen zur Fehlerausbreitung 205
4.3.2 Begrenzung der Fehlerausbreitung 209
4.3.3 Verfeinerung des Modellierungsgrades 214

4.4 Kriterien für Dominoeffektfreiheit 219

5. Beispiele dominoeffektfreier Prozeßsystemklassen 221

5.1 Wiederaufsetzpunktgesicherte globale Leseaufrufe 222

5.1.1 Charakterisierung vermeidbarer Zurücksetzung 222
5.1.2 Wirksamkeit globaler Leseaufrufsicherung 223

5.2 Wiederaufsetzpunktgesicherte Prozeßwechsel 232

5.2.1 Charakterisierung prozeßwechselbezogener Sicherungen 232
5.2.2 Wirksamkeit prozeßwechselbezogener Sicherungen 233

5.3 Beziehungsorientierte Sicherungen 236

5.3.1 Charakterisierung beziehungsorientierter Sicherungen 236
5.3.2 Wirksamkeit beziehungsorientierter Sicherungen 242

5.4 Transaktionsorientierte Sicherungen 247

5.4.1 Charakterisierung transaktionsorientierter Sicherungen 248
5.4.2 Wirksamkeit transaktionsorientierter Sicherungen 253

5.5 Rein azyklische Abhängigkeiten 259

5.5.1 Charakterisierung rein azyklischer Abhängigkeiten 259
5.5.2 Wirksamkeit rein azyklischer Abhängigkeiten 264

5.6 Berücksichtigung potentieller Fehlerverursacher 265

6. Ergebnisse themaverwandter Arbeiten 266

6.1 Kim's Prozeßablauf-Sicherungs-Schema 266

6.1.1 Kim's Prozeßsystem-Modell 266
6.1.2 Kim's Fehlerbehandlungs-Modell 266
6.1.3 Wesentliche Ergebnisse 271
6.1.4 Ergänzende Anmerkungen 275

6.2 Kant's Read-/Write-Modell 276

6.2.1 Kant's Prozeßsystem-Modell 276
6.2.2 Kant's Fehlerbehandlungs-Modell 277
6.2.3 Wesentliche Ergebnisse 280
6.2.4 Ergänzende Anmerkungen 282

6.2 Russell's Erzeuger-/Verbraucher-Modell 284

6.3.1 Russell's Prozeßsystem-Modell 284
6.3.2 Russell's Fehlerbehandlungs-Modell 284
6.3.3 Wesentliche Ergebnisse 288
6.3.4 Ergänzende Anmerkungen 289

7. Zusammenfassung mit Schlußbemerkungen 290

Literaturverzeichnis 293

Definierte Begriffe 299

Verwendete Symbole 308

Eingeführte Abkürzungen 310

1. Einleitung und Übersicht

Das Phänomen Dominoeffekt ist ein Problem, auf das man bereits vor über zehn Jahren aufmerksam wurde. Erste Ansätze zu seiner Behandlung findet man in [Hor74] und [Ran75] Seit dieser Zeit sind zahlreiche Beiträge erschienen, die sich ebenfalls mit diesem Thema beschäftigen. Daher gilt es an dieser Stelle die vorliegende Arbeit zu motivieren, die zu behandelnden Fragen abzugrenzen sowie an Beispielen zu veranschaulichen, die wesentlichen Ergebnisse vorzustellen und eine Übersicht zum Aufbau der Dissertation zu geben. Dies soll im folgenden geschehen.

1.1. Motivation

In Systemen nebenläufiger Prozesse, die gegenseitigen Wechselwirkungen unterliegen, besteht im Fehlerfall die Gefahr des Dominoeffektes. Informell erläutert bezeichnet man mit Dominoeffekt die zur Fehlerkorrektur notwendige Rücksetzung einer vorab nicht begrenzbaren Anzahl von Aktionen der Berechnungsfolge, die zum aktuellen, aber fehlerhaften Prozeßsystem-Zustand geführt hat.

Die Dominoeffekt-Gefahr kann nur dann zuverlässig ausgeschlossen werden, wenn vom System-Entwickler bereits während der Entwurfsphase eines Prozeßsystems, oder zumindest vom Betriebssystem her zur Laufzeit, besondere Vorkehrungen dagegen getroffen werden Sie besteht vor allem bei Verwendung von Wiederaufsetzpunkten zur Fehlerkorrektur, einer sehr gängigen Vorgehensweise.
In den letzten Jahren wurde deshalb dieses Problem in zahlreichen Aufsätzen aufgegriffen, beschrieben, analysiert, und es wurden verschiedentlich auch denkbare Lösungen aufgezeigt. Stellvertretend seien hier die Arbeiten von K.H. Kim, [Kim78a], [Kim78b], [Kim78c], [Kim78d], [Kim79], [Kim 80], D.L. Russell, [Rus76], [Rus77], [Rus80], S.K. Shrivastava und J. Banatre, [SB78], sowie die Dissertation von K. Kant und deren Veröffentlichung, [Kan81], [KS85], genannt.

Obwohl viele der veröffentlichten Arbeiten durchaus akzeptable Vorschläge zur Beseitigung des Dominoeffektes in Prozeßsystemen anbieten, unterliegen sie doch Einschränkungen und weisen Lücken auf. Insbesondere erscheinen in ihnen die folgenden Punkte nicht ausreichend allgemein oder nur unbefriedigend gelöst und daher änderungs- bzw. ergänzungswürdig:

(1) Es werden *nur spezielle Arten von Prozeßsystemen betrachtet*. In [Rus76], [Rus77], [Rus80] z.B. nur jene, die Nachrichten über besondere *send*- und *receive*-Befehle austauschen, in den Arbeiten [Kan81], [Kim78a], [Kim78b], [Kim78c], [Kim79], [Kim80], [SB78] wiederum nur solche, die über Monitore eines eingeschränkten Typs kommunizieren.
Eine Verallgemeinerung der gewonnenen Ergebnisse ist aus diesem Grunde nicht oder nur schwer möglich.

(2) Der *Grad der Formalisierung ist in der Regel gering*. Dadurch ist der Beweis von Aussagen zur Dominoeffektfreiheit von Prozeßsystemen überhaupt nicht oder nur in einer unbefriedigenden Art und Weise, die den Leser eher zu überreden als zu überzeugen versucht, möglich.
Typische Beispiele für Arbeiten der ersten Kategorie sind die Beiträge von K.H. Kim [Kim78a], [Kim78b], [Kim78c], [Kim79], [Kim80] und S.K Shrivastava/J. Banatre [SB78]. Einen Vertreter der zweiten Kategorie repräsentiert beispielsweise die Arbeit von K. Kant [Kan81].
Eine Ausnahme bilden die Arbeiten von D.L. Russell, [Rus76], [Rus77], [Rus80], deren Untersuchungen sich jedoch auf Erzeuger-/Verbraucher-Systeme mit Nachrichtenaustausch als Kommunikationshilfsmittel beschränken. Aber auch die Beiträge von D.L. Russell sind in den beiden, im folgenden genannten Punkten unbefriedigend.

Bezüglich des fehlenden Präzisierungsgrades fallen bei allen bisher erwähnten (und auch allen anderen bekannten) Arbeiten die folgenden beiden Punkte auf.

(3) Zentrale *Begriffe werden nur informell erläutert* und an Beispielen eingeführt, jedoch nicht explizit definiert. Dies gilt z.B. für die Begriffe Dominoeffekt, Dominoeffektfreiheit, fehlerhafter Zustand, fehlerfreier Zustand, berechnungstechnisch/syntaktisch konsistenter Zustand.

(4) Die *Modellbildung kommt zu kurz*. Das Fehlen eines umfassenden, formalen Fehlerbehandlungs-Modells und/oder Prozeßsystem-Modells erschwert (zusammen mit dem Fehlen der Präzisierung zentraler Begriffe) die formale Ableitung notwendiger Maßnahmen zur Fehlerbeseitigung. Denn diese können nur bei exakter Beschreibung der Menge der nach einem Fehler zurückzusetzenden Aktionen einer Berechnungsfolge und der Menge der potentiellen Fehlerverursacher, die beide, unter Zugrundelegung eines Fehlerbehandlungs-Modells, ein Maßstab für das Auftreten des Dominoeffektes sind, begründbar abgeleitet werden.

Die Unvollständigkeit bzw Ungenauigkeit bekannter Veröffentlichungen in den aufgezählten, wesentlich erscheinenden Punkten führte über eine detaillierte Kritik, [Bat83], der Arbeit von K. Kant, [Kan81], zur Erarbeitung der Aufgabenstellung für die vorliegende Dissertation, deren vorrangiges Ziel es ist, die skizzierten Einschränkungen und Lücken bekannter Arbeiten auf der Grundlage eines formalen Prozeßsystem- und Fehlerbehandlungs-Modells zu beseitigen bzw. zu schließen.

Nähere Einzelheiten zu den Schwerpunkten der Dissertation und zur Abgrenzung des Themas enthält der nächste Abschnitt.

1.2. Abgrenzung des Themas

Im Mittelpunkt dieser Arbeit steht die formale Untersuchung des bei der Fehlerbehandlung in Prozeßsystemen auftretenden Phänomens Dominoeffekt. Unter Zugrundelegung eines Prozeßsystem-Modells und eines entsprechenden Fehlerbehandlungs-Modells (Recovery-Modells) werden hinreichende und notwendige Bedingungen für das Auftreten des Dominoeffektes abgeleitet.
Je nach dem, ob im Recovery-Fall potentielle Fehlerverursacher berücksichtigt und zurückgesetzt werden oder nicht, wird gezeigt, daß der Dominoeffekt insbesondere durch eine nicht ausreichend auf die gegenseitigen Wechselwirkungen nebenläufiger Prozesse abgestimmte Strategie zum Anlegen prozeßlokaler Wiederaufsetzpunkte bzw., unter der Voraussetzung der Existenz adäquater Akzeptanztests, durch eine ungenügend koordinierte Vorgehensweise bei der Verifikation aktueller Prozeßsystemzustände begünstigt wird.

Die Ableitung dieser Ergebnisse ermöglicht die Identifizierung verschiedener dominoeffektfreier Prozeßsystem-Klassen und die Herleitung alternativer Vorgehensweisen zur Konstruktion dominoeffektfreier Systeme. Insbesondere wird in diesem Zusammenhang vorgeschlagen, in Analogie zum Deadlock-Problem Strategien zur Dominoeffekt-Vermeidung, -Verhinderung und -Erkennung mit anschließender Beseitigung abzuleiten.

Das Beschreiben hinreichender und notwendiger Bedingungen für das Auftreten des Dominoeffektes wird erst durch die Charakterisierung der Menge aller im Fehlerfall zurückzusetzenden Aktionen einer Berechnungsfolge ermöglicht. Diese Präzisierung beruht auf einer exakten Definition des Begriffes "berechnungstechnisch/syntaktisch konsistenter" und "fehlerfreier" Prozeßsystem-Zustand, dessen Formalisierung die Einführung und Präzisierung des Begriffes einer *Recoveryfunktion* ermöglicht.
Mit *Recoveryfunktion* wird eine Vorschrift bezeichnet, mit deren Hilfe ausgehend von einem fehlerhaften Prozeßsystem-Zustand ein berechnungstechnisch/syntaktisch konsistenter und in einem noch näher zu beschreibendem Sinne fehlerfreier Prozeßsystem-Zustand wiederhergestellt werden kann. Ihre Beschreibung stellt einen wesentlichen Beitrag dieser Arbeit dar, denn unter ihrer Verwendung können Ergebnisse zur Dominoeffektfreiheit von Prozeßsystemen insbesondere auch unter Berücksichtigung der zugrundeliegenden Fehlerbehandlungs-Strategien formal bewiesen werden.

Auf formale Nachweisbarkeit, aber auch auf Übertragbarkeit der gewonnen Ergebnisse wird in dieser Arbeit großer Wert gelegt. Sie werden deshalb, bis auf wenige Ausnahmen, ohne Einschränkung auf spezielle Synchronisations- und/oder Kommunikationsmechanismen abgeleitet. Auf diese Weise wird ihre allgemeine Anwendbarkeit gesichert.

In dieser Arbeit nicht behandelt werden Probleme der konkreten und effizienten Realisierung dominoeffektfreier Prozeßsysteme. Auf Lösungsansätze in diesem Bereich wird, soweit es sie gibt, in der folgenden, vier, für die Umsetzung von Konzepten zur Konstruktion dominoeffektfreier Prozeßsysteme wesentlich erscheinenden Punkte naher skizzierenden Diskussion, durch Herstellung von Bezügen zur vorhandenen Literatur hingewiesen.

Die vier Punkte, deren Berücksichtigung eine entscheidende Voraussetzung einer jeden Implementierung ist, können als Forderungen wie folgt formuliert und begründet werden:

(1) Die erste Forderung bezieht sich auf die *Begrenzung der Zahl aufzubewahrender Sicherungspunkte*:

Die Unterstützung der automatischen Fehlerkorrektur innerhalb eines Prozeßsystems setzt im allgemeinen Fall die Speicherung redundanter Information voraus. Dabei ist die Realisierbarkeit von der Zahl/Größe der aufzubewahrenden Sicherungspunkte abhängig.
Die Dominoeffektfreiheit eines Prozeßsystems kann allerdings, bei Verwendung von Sicherungspunkten zur Fehlerkorrektur, im allgemeinen Fall nur dann garantiert werden, wenn eine ständig wachsende Anzahl von Sicherungspunkten (die Prozeßsystem-Zustands-Historie) aufbewahrt wird. Aus diesem Grunde sind Überlegungen angebracht, die zum Ziel haben, die Zahl aufzubewahrender Sicherungspunkte zu begrenzen ohne dabei die Wahrscheinlichkeit für das Auftreten des Dominoeffektes wesentlich zu erhöhen. Derartige Fragestellungen werden z.B. in den Arbeiten von K.H. Kim [Kim78a], [Kim78b], [Kim78c] und D.L. Russell [Rus76] ansatzweise untersucht und diskutiert.

(2) Als zweite wichtige Forderung ist die *Ermittlung von optimalen Sicherungs-Intervallen* zu nennen:

Die Realisierung eines dominoeffektfreien Prozeßsystems macht nur dann Sinn, wenn das Systemverhalten nicht zu sehr durch den erforderlichen Sicherungsaufwand belastet und in seiner Funktion eingeschränkt wird. Deshalb muß es das Ziel einer jeden Implementierung sein, die Systembelastung zu minimieren.
Dieses Ziel kann durch Gegenüberstellung der Kosten für den Rücksetzaufwand verringernde Maßnahmen, wie z.B. das Anlegen von Sicherungspunkten, und der Kosten des zu erwartenden Rücksetzaufwandes erreicht werden, welche die Ermittlung eines optimalen Sicherungsintervalls ermöglicht.
Dieser Problemkreis wird in der Arbeit von N.F. Daouk, [Dao79], und in den Arbeiten von K.M. Chandy [Cha75a], [Cha75b], K.G. Shin und Y.H. Lee, [SL84], sowie J.W. Young [You74] aufgegriffen.

(3) Die Forderung drei nach der *Ableitung vollständiger und zuverlässiger Akzeptanztests* erscheint ebenfalls wesentlich:

Nur durch frühzeitiges Erkennen von Fehlern, kann ihre Ausbreitung eingedämmt und der zur Fehlerkorrektur erforderliche Rücksetzaufwand begrenzt werden. Mit Methoden zum Aufbau von Akzeptanztests und Möglichkeiten ihrer Nutzung zur Erreichung von Software-Fehlertoleranz beschäftigen sich z.B. die Aufsätze [And77], [Fab73] und [RC85] von D.M. Andrews, R.S. Fabry bzw. D.J. Richardson und L.A. Clarke. Eine Übersicht zu diesem Thema findet man in [AL81].
Die Erkennung einer Fehlersituation kann auch mittels der N-Versionen-Programmierung erfolgen, vgl. beispielsweise [Avi85], bei welcher eine Berechnung mehrfach auf jeweils unterschiedliche Art und Weise implementiert wird, und ein korrekter Zustand dadurch gekennzeichnet ist, daß mindestens eine vorgegebene Anzahl von Berechnungsalternativen zum gleichen Ergebnis kommt.

(4) Schließlich kann, analog zu zwei, die Forderung nach der *Bestimmung von optimalen Akzeptanztest-Intervallen* begründet werden:

Akzeptanztests sollen die frühzeitige Erkennung von Fehlern sichern und ihre Ausbreitung verhindern. Dieses Ziel ist nur durch ihre regelmäßige Ausführung erreichbar, was zu einer unerwünschten Änderung des Prozeßsystem-Verhaltens führen kann.
Durch Gegenüberstellung der Kosten für den zu erwartenden Rücksetzaufwand und der Kosten für die Durchführung von Akzeptanztests kann ein optimales Akzeptanztest-Intervall ermittelt werden.
Zu berücksichtigen ist allerdings, daß die Forderungen zwei und vier nicht voneinander unabhängig betrachtet werden dürfen. Daher erscheint die Erweiterung der Arbeit von N.F. Daouk, [Dao79], als eine gangbare Alternative für ein näheres Eingehen auf diesen Punkt.

Der in dieser Dissertation noch vorzustellende Modellansatz wäre auch geeignet, Möglichkeiten der Erfüllung der an eine Implementierung gestellten Anforderungen, wie der vier beschriebenen, näher zu analysieren. Eine in die vorliegende Arbeit direkt eingebettete Untersuchung dieses Themas würde jedoch den abgesteckten Rahmen sprengen. Dies ist der Grund, warum auf die in den Punkten (1) bis (4) aufgezeigten Problembereiche im folgenden nicht weiter eingegangen wird.

Soweit zur Abgrenzung des Inhalts. Der nächste Abschnitt führt in das zu behandelnde Thema anhand von Beispielen näher ein.

1.3. Einleitende Beispiele

Die zentrale Frage, auf welche diese Arbeit eine Antwort zu finden versucht, lautet:

> Unter welchen Bedingungen kann der Dominoeffekt sowohl bei *Nichtberücksichtigung* als auch bei *Berücksichtigung potentieller Fehlerverursacher* nicht auftreten?

oder anders formuliert:

> Durch welche Vorkehrungen kann die Zahl der im Fehlerfall, insbesondere bei Verwendung von Wiederaufsetzpunkten zur Fehlerkorrektur, zurückzusetzenden Aktionen sowohl unter *Nichtberücksichtigung* als auch unter *Berücksichtigung potentieller Fehlerverursacher* begrenzt werden?

Diese Fragestellung läßt sich am einfachsten anhand eines Beispiels veranschaulichen und erörtern

Angenommen zwei Prozesse P_1 und P_2 bearbeiten abwechselnd einen gemeinsamen Datenbestand, symbolisch durch die Variablen v_1 und v_2 dargestellt. Die auf dem gemeinsamen Datenbestand ausführbaren Operationen seien einfache, nichtunterbrechbare Lese- und Schreibaufrufe ($read(v_i)$, $write(v_i)$, i=1,2). Des weiteren sei angenommen, daß P_1 und P_2 Informationen auf dem folgenden Wege austauschen: P_1 verändert in unregelmäßigen Abständen den Wert der Variablen v_1, der, ebenfalls in unregelmäßigen Abständen, von Prozeß P_2 gelesen wird und dessen Ablauf beeinflußt. P_2 verändert die Variable v_2, deren Wert wiederum für P_1 von Bedeutung ist und daher von P_1 sporadisch gelesen wird. Bild 1.3/1 skizziert das Zusammenspiel zwischen den Prozessen P_1, P_2 beispielhaft.

Notationserläuterung zu den folgenden Bildern: Die zeitliche Reihenfolge (t) der Aktionen verläuft von oben nach unten (in Pfeilrichtung). Eine von P_i nach v_j gerichtete horizontale Kante deutet eine Schreibaktion, $write(v_j)$, des Prozesses P_i an; eine von v_j ausgehende Kante steht stellvertretend für ein $read(v_j)$ des Prozesses P_i, $i,j \in \{1,2\}$.
Die Aktion $a_{1,1}$ ist beispielsweise eine Leseaktion ($read(v_2)$) und die Aktion $a_{2,1}$ ist eine Schreibaktion ($write(v_2)$), wobei $a_{2,1}$ zeitlich vor $a_{1,1}$ ausgeführt wurde.
Im Bild 1.3/1 sind nur die für die anschließende Diskussion wesentlichen Aktionen der Prozesse P_1 und P_2 benannt. Die Änderung eines Variablenwertes wird durch Änderung der Schraffierung angedeutet.

Im Rahmen einer Diskussion zweier Fälle wird im folgenden gezeigt, daß

Fehlerkorrektur zur Zurücksetzung der Prozesse P_1 und P_2 in den Anfangszustand und damit zur Zurücksetzung einer nicht vorab berechnungsunabhängig begrenzbaren Anzahl von Aktionen führen kann.

Im ersten Fall wird angenommen, daß im in Bild 1.3/1 skizzierten Beispiel weder für Prozeß P_1 noch für Prozeß P_2 Wiederaufsetzpunkte definiert werden. Bei Zugrundelegung einer Fehlerbehandlungs-Strategie, die auf Rücksetzung fehlerhafter Prozesse zu einem früheren berechnungstechnisch/syntaktisch konsistenten und fehlerfreien Prozeßzustand beruht, erfordert deshalb die Korrektur eines nach Aktion $\alpha_{1,7}$ oder $\alpha_{2,8}$ im Prozeß P_1 bzw. P_2 auftretenden Fehlers die Zurücksetzung beider Prozesse in den als (einzigen) gesichert und berechnungstechnisch/syntaktisch konsistent sowie fehlerfrei vorausgesetzten Anfangszustand. Denn durch die Zurücksetzung von P_1 (bzw. P_2) wird P_2 (bzw. P_1) Information entzogen, nämlich die durch $\alpha_{1,2}$, $\alpha_{1,5}$ und $\alpha_{1,7}$ (bzw. $\alpha_{2,1}$, $\alpha_{2,3}$, $\alpha_{2,4}$ und $\alpha_{2,7}$) veränderten Variablenwerte von v_1 (bzw. v_2), die den Ablauf von P_2 (bzw. P_1) nach Annahme beeinflußt haben.
Diese Abhängigkeit zwischen P_1 und P_2 veranschaulicht Bild 1.3/2.

Auch die Einführung von Wiederaufsetzpunkten muß im Fehlerfall nicht vor Zurücksetzung der Prozesse P_1 und P_2 in den Anfangszustand schützen, wie Fall zwei zeigt.

In Bild 1.3/3 wurden in die Prozeßabläufe von P_1 und P_2 Wiederaufsetzpunkte $WP_{i,j}$ eingefügt, $i \in \{1,2\}$, $j \in \{1,2,3,4\}$ und $j \in \{1,2,3\}$ für P_1 bzw. P_2. Falls nun z.B. Aktion $\alpha_{1,7}$ aufgrund eines Fehlers rückgängig gemacht werden müßte, könnte dies durch Rücksetzen des Prozesses P_1 zum Wiederaufsetzpunkt $WP_{1,4}$ geschehen. Durch das Zurücksetzen von Aktion $\alpha_{1,7}$ würde Prozeß P_2 die über Aktion $\alpha_{2,8}$ erhaltene Information entzogen. $\alpha_{2,8}$ müßte deshalb korrigiert und damit P_2 zum Wiederaufsetzpunkt $WP_{2,3}$ zurückgesetzt werden. Dies wiederum würde eine Korrektur der Aktion $\alpha_{1,6}$ und damit auch die Korrektur des Zustandes von Prozeß P_1 erforderlich machen. P_1 müßte deshalb zum Wiederaufsetzpunkt $WP_{1,3}$ zurückgesetzt werden usw.
Diese Kettenreaktion, die als "klassischer" Dominoeffekt bezeichnet werden kann und schematisch in Bild 1.3/4 festgehalten ist, würde schließlich zur Zurücksetzung der Prozesse P_1 und P_2 in den Anfangszustand führen.

Eine analoge Kettenreaktion würde auch die Zurücksetzung einer Aktion des Prozesses P_2 verursachen. In Bild 1.3/5 ist dieser Sachverhalt beispielhaft für Aktion $\alpha_{2,8}$ eingezeichnet.

Den beiden skizzierten Fällen (vgl. zu Fall 1 Bild 1.3/2 bzw. zu Fall 2 die Bilder 1.3/4, 5) ist gemeinsam, daß im Fehlerfall, eine auf Prozeßrücksetzung beruhende Fehlerbehandlungs-Strategie vorausgesetzt, eine potentiell nicht vorab

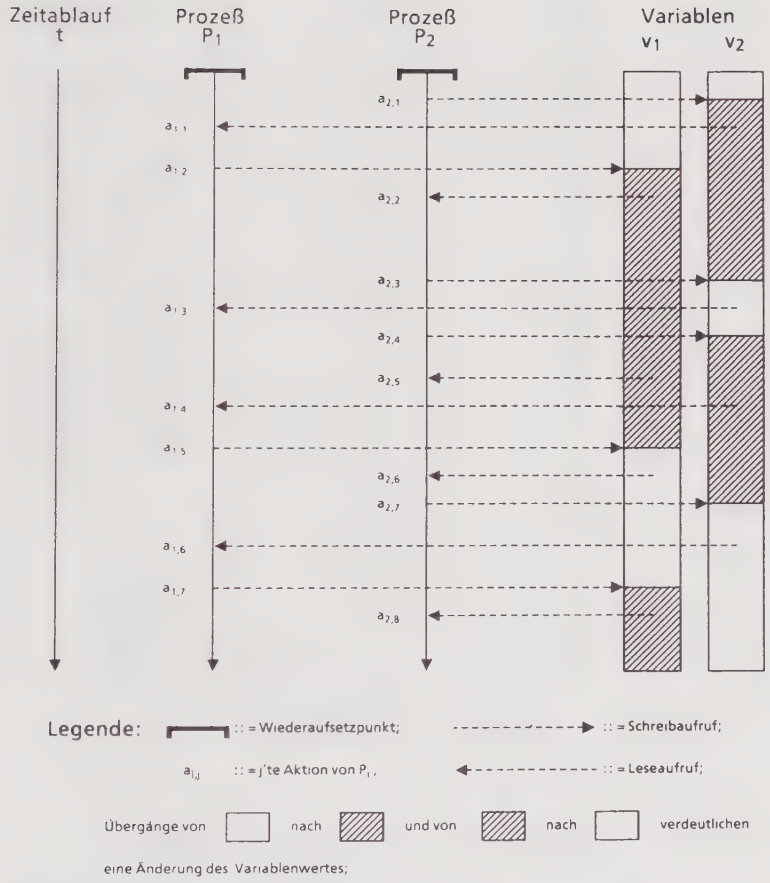
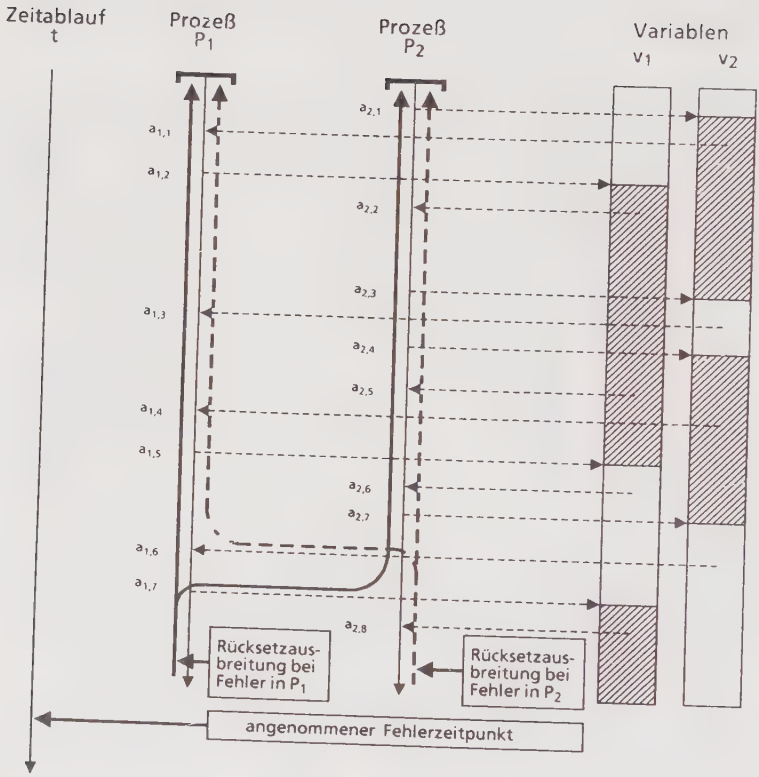
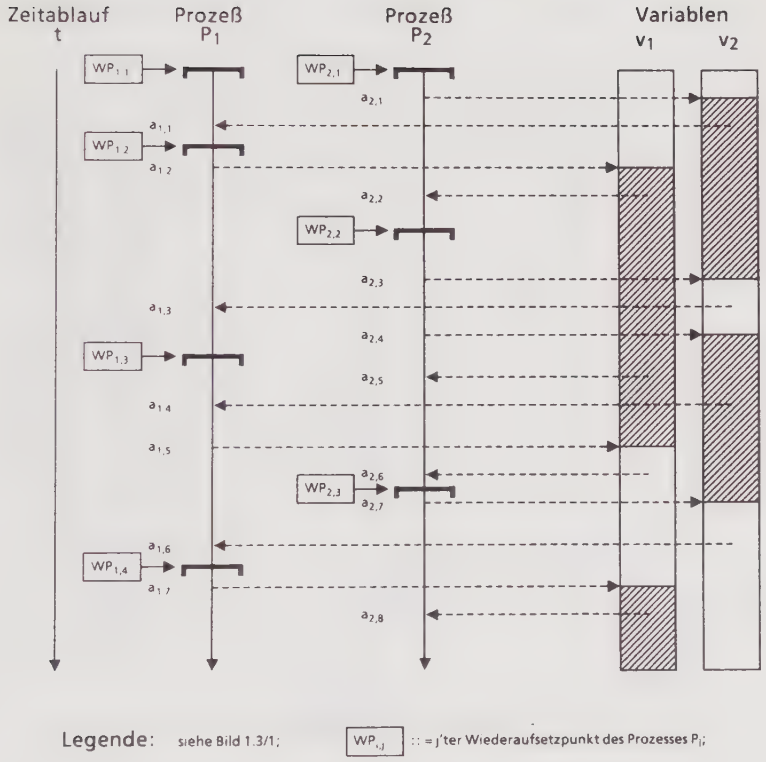


Bild 1.3/1: Zusammenspiel zwischen den Prozessen P_1 und P_2



Legende: siehe Bild 1.3/1;

Bild 1.3/2: Fehlerbehandlungsbezogene Abhängigkeiten zwischen P_1 und P_2



Legende: siehe Bild 1.3/1; $WP_{i,j}$ :: = j'ter Wiederaufsetzpunkt des Prozesses P_i;

Bild 1.3/3: Mehrfach wiederaufsetzpunktgesicherte Prozeßabläufe von P_1 und P_2

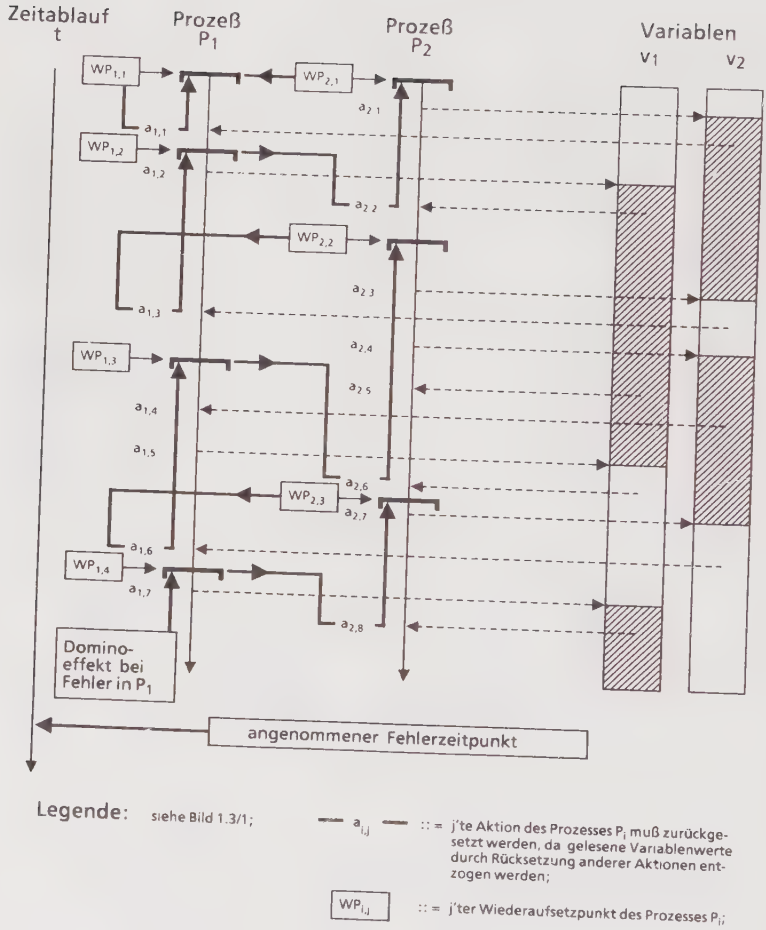


Bild 1.3/4: Klassischer Dominoeffekt von P_1 ausgehend

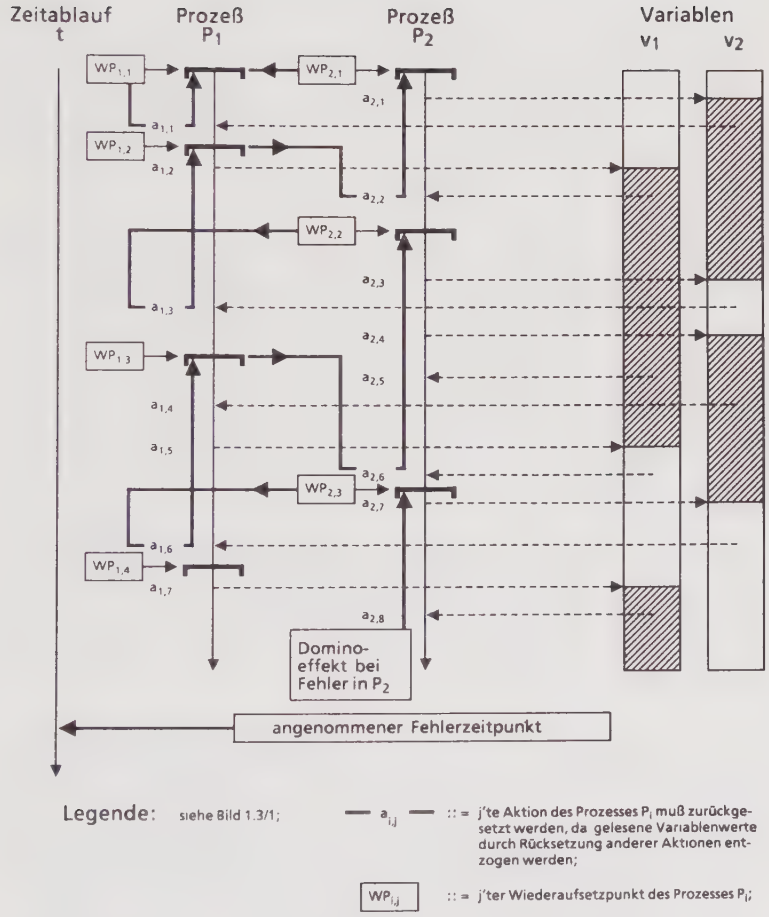


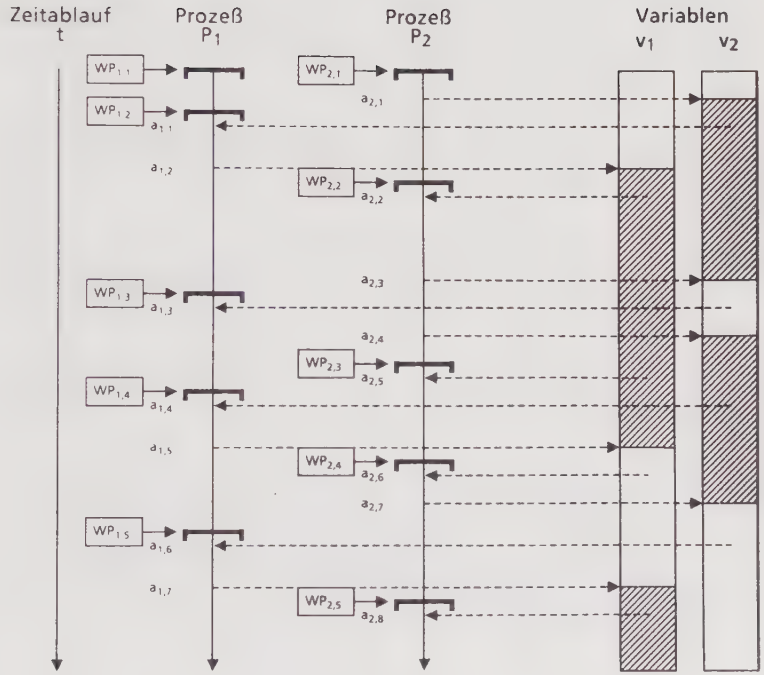
Bild 1.3/5: Klassischer Dominoeffekt von P_2 ausgehend

berechnungsunabhangig begrenzbare Anzahl von Aktionen zum Zwecke der Wiederherstellung eines konsistenten Prozeßsystem-Zustandes zurückgesetzt werden muß. Aus diesem Grunde wird in dieser Arbeit in beiden Fällen, abweichend von der gängigen Bezeichnungsweise, von der Gefahr eines Dominoeffektes gesprochen, und beide Situationen werden in die Betrachtungen miteinbezogen In der Fachliteratur wird in der Regel nur ein zu Fall zwei analoger Sachverhalt, der als *nicht berechnungsunabhängig begrenzbare zyklische Zurücksetzung* bezeichnet wird, mit dem Begriff Dominoeffekt in Verbindung gebracht.

Obwohl, wie anhand der Bilder 1.3/4 und 5 gezeigt wurde, die Einführung von Prozeßzustands-Sicherungspunkten (Wiederaufsetzpunkten) nicht zwangsläufig die Gefahr eines Dominoeffektes ausschließt, kann, was noch zu beweisen sein wird, durch Beachtung verschiedener Grundsätze in der im vorgegebenen Beispiel beschriebenen Sachlage, die (gleichzeitige) Zurücksetzung beider Prozesse P_1 und P_2 in den Anfangszustand im Fehlerfall (zumindest bei Einfachfehlern) vermieden werden.
Bild 1.3/6 zeigt eine der Möglichkeiten. Durch Sicherung des aktuellen Prozeßzustandes unmittelbar vor jedem sich auf die prozeßgemeinsamen Variablen v_1 und v_2 beziehenden Leseaufruf kann der Einfluß der Zurücksetzung von Aktion $a_{2,8}$ auf Prozeß P_1 eingedämmt werden, und die Zurücksetzung von P_1 zum Wiederaufsetzpunkt $WP_{1,5}$ führt lediglich zur Zurücksetzung von Prozeß P_2 zum Wiederaufsetzpunkt $WP_{2,5}$, aber zu keinen weiteren Rücksetz-Aktivitäten.

Die Erörterung des zu behandelnden Problemkreises wurde bisher ohne Berücksichtigung potentieller Verursacher eines Fehlers geführt. Da im Rahmen der vorliegenden Arbeit auch auf dieses Thema näher eingegangen wird, soll an dieser Stelle kurz erläutert werden, was mit der Berücksichtigung potentieller Fehlerverursacher und mit der Begrenzung des Rücksetzaufwandes unter Berücksichtigung potentieller Fehlerverursacher gemeint ist.

Angenommen in der Situation von Bild 1.3/6 würde sich herausstellen, daß Aktion $a_{2,8}$ einen Fehler verursacht hat und deshalb zurückgesetzt werden muß. Im Falle der Nichtberücksichtigung potentieller Fehlerverursacher, d.h. der impliziten Annahme, die an Aktion $a_{2,8}$ übergebene Information v_1 sei korrekt gewesen, reicht zur Fehlerkorrektur die Rücksetzung von Prozeß P_2 zum Wiederaufsetzpunkt $WP_{2,5}$ aus. Geht man von dieser impliziten Annahme nicht aus, so muß, falls das Ergebnis der Aktion $a_{1,7}$ bis zum Fehlerzeitpunkt nicht explizit durch einen zuverlässigen Akzeptanztest verifiziert wurde bzw nachträglich verifiziert werden kann, auch das Ergebnis der Aktion $a_{1,7}$ in Frage gestellt und Prozeß P_1 zum Wiederaufsetzpunkt $WP_{1,5}$ zurückgesetzt werden. Durch analoge Überlegungen kann schließlich die Forderung nach Zurücksetzung der Aktionen $a_{2,7}$, $a_{1,5}$, $a_{2,4}$, $a_{1,2}$, $a_{2,1}$ und damit die Forderung nach Zurücksetzung der Prozesse P_1, P_2 in den Anfangszustand begründet werden, falls die Korrektheit der Prozeßzustände und des Informationsflusses im Fehlerfall nicht entsprechend



Legende: siehe Bild 1.3/1 und 3;

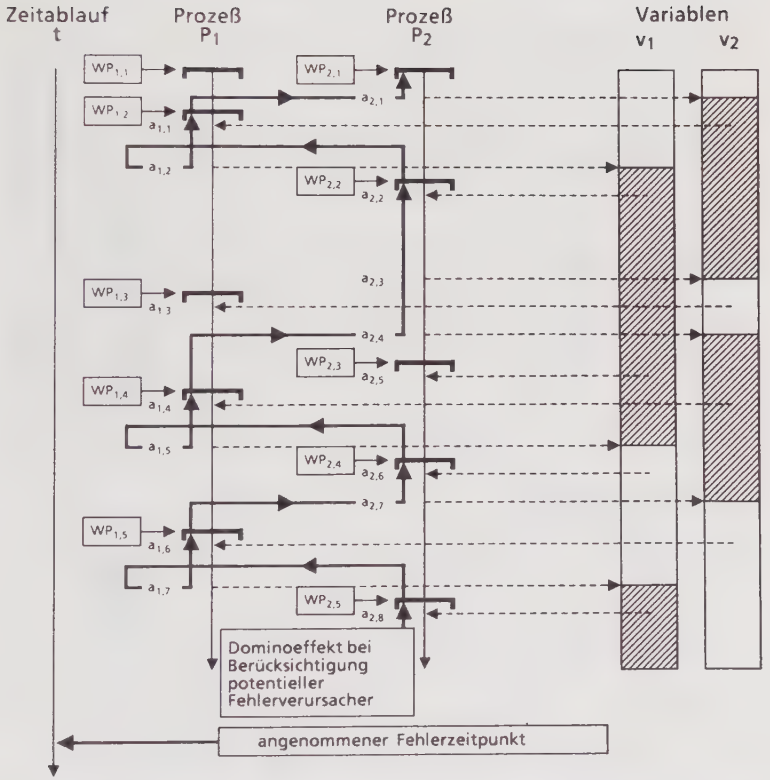
Bild 1.3/6: Begrenzung der Rücksetzausbreitung durch Leseaufrufsicherung

nachgewiesen werden kann.
Anhand dieses Beispiels sieht man, daß das Problem des Dominoeffektes auch entstehen kann, in dem man bei einem dominoeffektfreien Prozeßsystem von der impliziten Annahme der Korrektheit der zwischen Prozessen übergebenen Information abgeht und die Richtigkeit in Frage stellt, siehe Bild 1.3/7.

Wie noch zu zeigen sein wird, kann durch gezieltes Setzen adäquater Akzeptanztests innerhalb der einzelnen Prozesse eines Prozeßsystems die Gefahr des Dominoeffektes auch bei Berücksichtigung potentieller Fehlerverursacher beseitigt werden.
Beispielsweise kann durch Einführung von Akzeptanztests im Anschluß an jeden Schreibaufruf, der die prozeßgemeinsamen Variablen v_1 und v_2 verändert, die gleichzeitige Zurücksetzung der Prozesse P_1 und P_2 in den Anfangszustand, vgl. Bild 1.3/7, selbst bei Berücksichtigung potentieller Fehlerverursacher, vermieden werden.

In der in Bild 1.3/8 dargestellten Sachlage kann Aktion $a_{2,8}$ das Ergebnis von Aktion $a_{1,7}$ nicht mehr in Frage stellen, da dessen Korrektheit durch den erfolgreichen Abschluß des Akzeptanztests $AT_{1,3}$ bereits nachgewiesen wurde.

Die Frage wie Akzeptanztests in bestehende Prozeßsysteme einzufügen sind wird in dieser Arbeit insbesondere deshalb behandelt, da sie im gewissen Sinne als dual zum Problem des dominoeffekt-eindämmenden Setzens von Wiederaufsetzpunkten angesehen werden kann. Voraussetzung für ihre Beantwortung ist natürlich die Existenz und Zuverlässigkeit adäquater Akzeptanztests, welche im konkreten Fall die implizite Annahme, die Menge der potentiellen Fehlerverursacher sei leer, bei derartigen Betrachtungen ersetzt.



Legende: siehe Bild 1.3/1 und 3; — $a_{i,j}$ — :: = j'te Aktion des Prozesses P_i muß zurückgesetzt werden, da sie potentieller Fehlerverursacher ist;

Bild 1.3/7: Fehlerausbreitungsbedingter Dominoeffekt

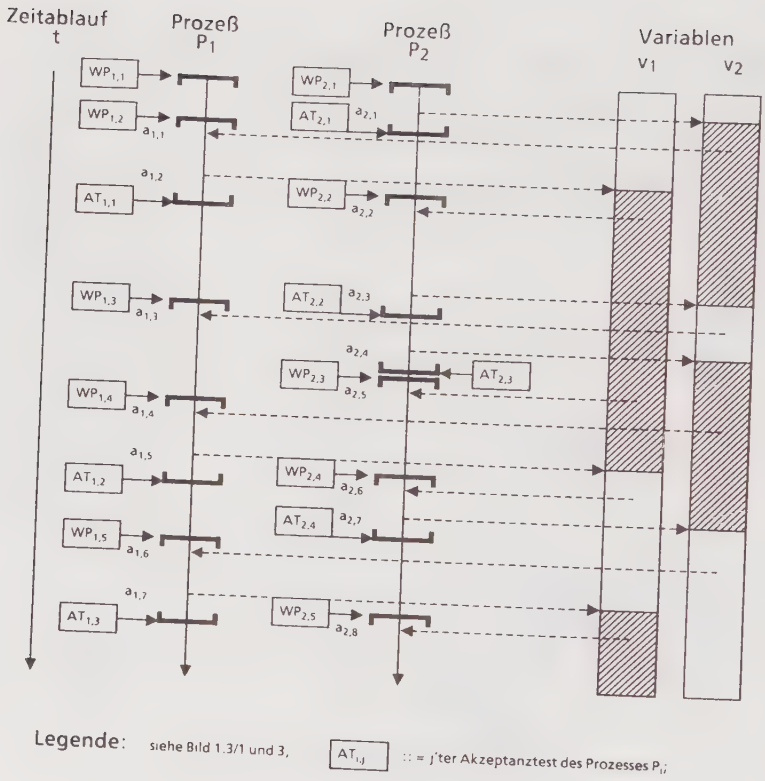


Bild 1.3/8: Begrenzung der Fehlerausbreitung durch Schreibaufrufverifizierung

1.4. Wesentliche Ergebnisse

Die wesentlichen Ergebnisse dieser Arbeit lassen sich stichpunktartig wie folgt charakterisieren:

(1) Unter Benutzung eines formalen Prozeßsystem-Modells wird, nach vorheriger Erläuterung und Festlegung zentraler Grundbegriffe und Bezeichnungsweisen, u.a. auch des Begriffes konsistenter Prozeßsystem-Zustand, das *Konzept einer Recoveryfunktion* eingeführt.

(2) Es wird gezeigt, daß das Aussehen einer Recoveryfunktion, mit ihr kann ein fehlerhafter Prozeßsystem-Zustand in einen konsistenten Prozeßsystem-Zustand überführt werden, durch das einem Prozeßsystem zugrundeliegende Fehlerbehandlungs-Modell geprägt ist.

(3) Mit Hilfe der Recoveryfunktion läßt sich im Fehlerfall die Menge derjenigen Aktionen begründbar ableiten, ohne deren Zurücksetzung eine Fehlerkorrektur nicht möglich ist.
Die Recoveryfunktion stellt somit einen Zusammenhang zwischen Prozeßsystem-Modell, Fehlerbehandlungs-Modell und den Zielen Dominoeffekt-Gefahr-Begrenzung/-Beseitigung dar.

(4) Einen weiteren wesentlichen Aspekt stellt die Ableitung notwendiger und hinreichender Bedingungen für das Auftreten des Dominoeffektes dar. Hierbei werden zwei Teilprobleme unterschieden: Das Problem der Rücksetzausbreitung und das der Fehlerausbreitung, die beide, voneinander unabhängig, zu einem Dominoeffekt führen können.

(5) Das Aufzeigen der Ursachen des Dominoeffektes ermöglicht schließlich dominoeffektfreie Prozeßsystem-Klassen zu identifizieren sowie alternative Strategien zur Dominoeffekt-Gefahr-Begrenzung/-Beseitigung ausfindig zu machen und zu diskutieren.

(6) Die wesentlichen aus der Literatur bekannten Ergebnisse zur Dominoeffekt-Freiheit von Prozeßsystemen, wie sie vor allem in den Beiträgen von K. Kant, [Kan81], von K.H. Kim, [Kim78a], [Kim78b], [Kim78c], [Kim79], [Kim80], D.L. Russell, [Rus76], [Rus77], [Rus80], und von S.K. Shrivastava/J. Banatre, [SB78], aufgezeigt, aber in der Regel nicht oder nur unzureichend nachgewiesen wurden, werden bestätigt, in wichtigen Punkten verallgemeinert und zusammen mit neu hinzukommenden Resultaten formal bewiesen.

1.5. Übersicht zum Aufbau

Die vorliegende Dissertation ist in sieben Kapitel gegliedert. Diese bauen inhaltlich wie folgt aufeinander auf:

Nach Abschluß der *Einleitung und Übersicht* wird im Kapitel zwei das den Untersuchungen zugrundeliegende formale Modell, unterteilt nach *Prozeßsystem- und Fehlerbehandlungs-Modell* beschrieben und in allen wesentlich erscheinenden Punkten detailliert erläutert.
Grundlegende Vereinbarungen werden im Kapitel drei getroffen. Sie umfassen die Einführung elementarer Begriffe, die Präzisierung wichtiger Konzepte (incl. des der Recoveryfunktion) und Überlegungen, sowie die Integration wesentlicher Annahmen und vorausgesetzter Systemeigenschaften des Fehlerbehandlungs-Modells durch Definition der Menge der im Fehlerfall zurückzusetzenden Aktionen mit und ohne Berücksichtigung potentieller Fehlerverursacher.
Eine *Charakterisierung des Dominoeffektes* erfolgt im Kapitel 4, unterteilt nach den beiden Hauptursachen Rücksetzausbreitung und Fehlerausbreitung. Die Charakterisierung des Dominoeffektes stellt die Basis für die *Identifizierung dominoeffektfreier Prozeßsystem-Klassen* im fünften Kapitel dar.
Auf der Grundlage der im Kapitel vier und fünf abgeleiteten Ergebnisse können *Strategien zur Dominoeffekt-Gefahr-Begrenzung/-Beseitigung* diskutiert werden. Ziel dieser Überlegungen ist es, in Analogie zum Deadlockproblem Vorgehensweisen zur Dominoeffekt-Gefahr-Vermeidung, -Verhinderung oder zumindest rechtzeitiger Erkennung und anschließender Beseitigung ausfindig zu machen.
Im Kapitel sechs werden schließlich *Bezüge zu themaverwandten Arbeiten* hergestellt, und es wird auf diese Weise gezeigt, daß die in ihnen abgeleiteten wesentlichen Resultate durch die in der vorliegenden Dissertation bewiesenen Sätze abgedeckt werden.
Eine *Zusammenfassung mit Schlußbemerkungen* rundet die Ausarbeitung ab.

Nähere Angaben zu der benutzten Literatur und den im Text eingestreuten Referenzen enthält ein ausführliches *Literaturverzeichnis*. Es ist, ebenso wie eine alphabetisch geordnete *Liste der definierten Begriffe*, eine *Aufstellung der verwendeten Symbole* und eine übersichtliche tabellarische *Zusammenstellung der eingeführten Abkürzungen*, am Ende der Arbeit angefügt.

2. Zugrundeliegendes Modell

Wie bereits in den einleitenden Abschnitten erläutert, kann eine Untersuchung der im ersten Kapitel angerissenen Fragestellungen nur unter Zugrundelegung eines formalen Modells präzise durchgeführt werden. Im vorliegenden Fall läßt sich ein derartiges Modell in zwei Hauptbestandteile, ein *Prozeßsystem-* und ein *Fehlerbehandlungs-Modell*, zerlegen. Beide werden im folgenden in der genannten Reihenfolge beschrieben.

2.1. Prozeßsystem-Modell

Das verwendete Prozeßsystem-Modell ist an eine Formalisierung von R.J. Lipton, [Lip74], angelehnt (vgl. auch [Bat82], [Hof84]). Die Darstellung orientiert sich weitgehend an den in [Bat82] und [Hof84] gewählten Formen. Zitate aus den letztgenannten beiden Arbeiten sind in diesem Abschnitt nicht ausdrücklich gekennzeichnet.

Nach der Einführung grundlegender Begriffe wird auf die Unterscheidung verschiedener Aktionentypen und Prozeßbeziehungen sowie auf die Modellierung von Abhängigkeiten zwischen Prozessen und der darauf aufbauenden Behandlung unterschiedlicher Prozeßsystemklassen eingegangen.

2.1.1. Formalisierung wesentlicher Begriffe

Die drei wesentlichen Elemente des Prozeßsystem-Modells von Lipton, [Lip74], sind die drei Begriffe *Aktion*, *Prozeßsystem* und *Berechnungsablauf*. Darüberhinaus sind elementare Funktionen und Relationen zu berücksichtigen, die auf diesen Begriffen basieren und wichtige mit ihnen häufig verbundene Eigenschaften widerspiegeln. Die Präzisierung dieser Begriffe, Funktionen und Relationen ist Gegenstand der nachfolgenden Ausführungen. In Abschnitt 2.1.1.4 werden die eingeführten Bezeichnungen an einem Beispiel veranschaulicht.

2.1.1.1. Grundlegende Vereinbarungen

Werden Namen für neue Begriffe oder Funktionen eingeführt, deren Bedeutung durch schon definierte erklärt wird, so wird dies durch die folgende Schreibweise zum Ausdruck gebracht: *definiendum* : = *definiens*. Werden neue Prädikate oder Aussagen als Abkürzungen für schon vorhandene eingeführt, so wird dies durch *definiendum* ::<=> *definiens* zum Ausdruck gebracht. Das Zeichen ":=" wird in dieser Arbeit analog

der ALGOL 60-Konvention als *Wertzuweisungssymbol* verwendet.
Mit $\mathbf{Z}$ wird die Menge der *ganzen Zahlen*, mit $\mathbf{N}$ die Menge der *natürlichen Zahlen ohne die Null* und mit $\mathbf{N}_0$ die Menge der *natürlichen Zahlen mit der Null* bezeichnet.

Sei $D = D_1 \times \ldots \times D_n$ eine Menge. $(x_1, \ldots, x_n)$ heißt *typisches Element* von D, wenn jedes x_i eine Variable mit Wertebereich D_i ist $(i \in \{1, \ldots, n\})$.
Ist $y = (a_1, \ldots, a_n) \in D$ und $(x_1, \ldots, x_n)$ typisches Element von D, so wird die Schreibweise $x_i[y]$ als Bezeichnung für die Komponente a_i von y vereinbart.

Beispiel 2.1.1.1/1: Sei $D = \{1, 2, 3\} \times \{1, 2, 3\}$, (x_1, x_2) typisches Element von D und $y = (1, 3)$. Dann ist $x_1[y] = 1$ und $x_2[y] = 3$. $x_i[y]$ bezeichnet also die Projektion von y auf die i-te Komponente.

Sei $D = D_1 \times \ldots \times D_n$ und $(x_1, \ldots, x_n)$ typisches Element von D. Für eine *Abbildung* α von D nach D wird die folgende Notation eingeführt:

$$\alpha \ ::= \ \textit{when} \ B_\alpha(x_1, \ldots, x_n) \ \textit{do} \ (x_1, \ldots, x_n) := G_\alpha(x_1, \ldots, x_n); \ \textit{od};$$

Hierbei ist B_α ein Prädikat und G_α eine Funktion. Für α gilt:

$$\alpha(x_1, \ldots, x_n) = \begin{cases} G_\alpha(x_1, \ldots, x_n) & B_\alpha(x_1, \ldots, x_n) \\ & \textit{falls} \\ (x_1, \ldots, x_n) & \neg B_\alpha(x_1, \ldots, x_n) \end{cases}$$

Eine Abbildung α wird im weiteren als *Aktion* bezeichnet.

Beispiel 2.1.1.1/2: Sei $D = \{1, 2, 3\} \times \{1, 2, 3\}$, (x_1, x_2) typisches Element von D, und die Aktion α sei definiert durch:

$$\alpha \ ::= \ \textit{when} \ x_1 < x_2 \ \textit{do} \ x_1 := x_2; \ x_2 := x_1; \ \textit{od};$$

Dann besitzt α an der Stelle (2,3) den Wert (3,2).

Bemerkung: Man beachte, daß die Anweisungen hinter "*do*" einer *parallelen Wertzuweisung* entsprechen und daher nicht sequentiell, sondern gleichzeitig auszuführen sind. Des weiteren wird eine *Aktion als zeitlich unteilbar betrachtet*.
Im folgenden werden innerhalb einer Aktion Zuweisungen der Form "$x := x$" weggelassen.

2.1.1.2. Prozeßsystem-Begriff

Die Grundlage des Modells von Lipton, [Lip74], bildet der Begriff des *Prozeßsystems*.

Definition 2.1.1.2/1: $P=(A,D,w)$ heißt *Prozeßsystem* genau dann, wenn gilt:

(1) $D = D_1 \times \ldots \times D_n \times D_{V_r}$ ist endliches kartesisches Produkt mit typischem Element $(bz_1, \ldots, bz_n, V_r)$ und heißt Zustandsmenge. $w \in D$ ist der Anfangszustand des Systems und A ist eine endliche Menge von Aktionen.

(2) Mit P implizit verbunden sind die beiden Funktionen bz und adr.

$$bz: A \to \{1, \ldots, n\} \text{ und } adr: A \to \bigcup_{i=1}^{n} D_i$$

bz ordnet jede Aktion einem Prozeß (Befehlszähler) zu; adr ordnet jeder Aktion eine Aktionsadresse zu, wobei $adr(a) \in D_{bz(a)}$ ist.

(3) Eine Aktion $a \in A$ ist eine Abbildung von D nach D der folgenden Struktur:

$$when\ \ bz_{bz(a)}=adr(a)\ \&\ B_a(V_r)\ \ do\ \ bz_{bz(a)} := N_a(V_r);\ \ V_r := F_a(V_r);\ \ od;$$

Der Teil $(bz_{bz(a)}=adr(a)\ \&\ B_a(V_r))$ wird auch als Ausführungsbedingung (Bedingungsteil) von a bezeichnet. Die Anweisungsfolge zwischen "*do*" und "*od*" als Operationsteil.

(4) Für zwei Aktionen $a_i, a_j \in A$ gilt:

$$[adr(a_i)=adr(a_j)\ \&\ bz(a_i)=bz(a_j)] \Rightarrow a_i=a_j,$$

d.h. eine Aktion ist durch ihre Prozeßzugehörigkeit und Aktionsadresse eindeutig bestimmt.

Informell kann man sich unter einem Prozeßsystem eine Menge von n sequentiellen Prozessen vorstellen, die unter Berücksichtigung bestimmter Restriktionen parallel ausführbar sind. A ist der Programmtext der Prozesse. $D_1, \ldots, D_n$ sind die Adressräume und $bz_1, \ldots, bz_n$ die Befehlszähler der n Prozesse. In der Komponente D_{V_r} von D sind die Wertebereiche sämtlicher (restlicher) Variablen von P zusammengefaßt und V_r enthält ihre Namen. Durch die Elemente aus D kann der Zustand eines Prozeßsystems vollständig beschrieben werden. Die Ausführung einer Aktion eines Prozesses kann diesen Zustand verändern.

Die in der Beschreibung einer Aktion vorkommenden Bestandteile können folgendermaßen interpretiert werden. Mit der Funktion adr kann die Adresse einer Aktion $a \in A$ berechnet werden. bz erlaubt die Bestimmung der Prozeßzugehörigkeit

einer Aktion $bz_{bz(\alpha)}=adr(\alpha)$ im Bedingungsteil einer Aktion α bedeutet: α kann frühestens dann ausgeführt werden, wenn der Befehlszählerstand des Prozesses $bz(\alpha)$ mit $adr(\alpha)$ übereinstimmt. Durch das Prädikat $F_\alpha(V_r)$ können noch zusätzliche Bedingungen an die Ausführbarkeit einer Aktion geknüpft werden. F_α kann ergänzend gefordert werden, daß α nur dann ausführbar sein soll, wenn bestimmte Variablen positive Werte besitzen.
Durch die Funktion N_α wird die Nachfolgeraktion von α im Prozeß $bz(\alpha)$ bestimmt, d.h. N_α modifiziert den Befehlszähler $bz_{bz(\alpha)}$ (und zwar nur diesen Befehlszähler) des Prozesses, zu dem α gehört.
Mit der Funktion F_α können Berechnungen durchgeführt und den Variablen in V_r neue Werte zugewiesen werden.

2.1.1.3. Berechnugs-Begriff

In einem Prozeßsystem werden Berechnungen betrachtet, die als *Abläufe* bezeichnet werden und den Systemzustand ändern können.

Definition 2.1.1.3/1: Sei $P=(A,D,u)$ ein Prozeßsystem. Jede endliche Folge α von Aktionen aus A wird als *Ablauf* bezeichnet.

Notationsvereinbarung: Endliche Aktionenfolgen werden im folgenden mit kleinen griechischen Buchstaben ($\alpha, \beta, \gamma, \ldots$) bezeichnet. λ ist die *leere Folge*.
Ist α ein Ablauf, so bezeichnet $\alpha[i]$ das i-te Element der Aktionenfolge α, α_i das Anfangsstück bis einschließlich dem i-ten Element. $\alpha[0]:=\lambda$, $\alpha_0:=\lambda$.
A^* repräsentiert die Menge aller Abläufe über der Aktionenmenge A (die Menge aller Wörter über dem Zeichenvorrat A). Die Konkatenation zweier Folgen α und β in A^* wird als Folge $\alpha\beta$ dargestellt, d.h. auf die explizite Angabe eines Konkatenations-Operators zwischen den Folgen α und β wird verzichtet.
$|\alpha|$ gibt die Länge der Folge α an. $W(\alpha)$ wird verwendet als Abkürzung für die Menge aller in α vorkommenden Aktionen $\alpha[i]\in\{\alpha[1], \ldots \alpha[|\alpha|]\}$, die über den in eckigen Klammern stehenden Index unterschieden werden, auch wenn sie bezogen auf die Menge A identische Aktionen darstellen. $|W(\alpha)|$ gibt die Kardinalität der Menge $W(\alpha)$ an.

Die Wirkung eines Ablaufes auf den Systemzustand eines Prozeßsystems $P=(A,D,u)$ wird durch eine Funktion val_P beschrieben.

Definition 2.1.1.3/2: Sei $P=(A,D,u)$ ein Prozeßsystem. Die Funktion val_P, die Abläufen aus A^* Zustände aus D zuordnet, wird definiert durch:

(1) $val_P(\lambda) = w$,

(2) Ist α ein Ablauf und a eine Einzelaktion, dann soll gelten:

$$val_P(\alpha a) = a(val_P(\alpha)).$$

Um Beziehungen zwischen einzelnen Prozessen eines Prozeßsystems P einfach beschreiben zu können, werden im Liptonschen Modell des weiteren die Relation $proc_P$ sowie die Funktionen bef_P und lf_P eingeführt.

Definition 2.1.1.3/3: Sei $P=(A,D,w)$ ein Prozeßsystem, $(bz_1, ..., bz_n, V_r)$ typisches Element von D und $\alpha \in A^*$ ein Ablauf. Dann werden die Relation $proc_P \subset A \times A$, die Funktion $bef_P: A^* \to P(A)$ und die Funktion $lf_P: A^* \to P(A)$ definiert durch (wobei $P(A)$ in diesem Zusammenhang die Potenzmenge bezeichnet):

(1) $proc_P(a_i,a_j) \ ::<=> \ a_i,a_j \in A \ \& \ bz(a_i)=bz(a_j)$,

(2) $bef_P(\alpha) \ ::= \ \{a \mid a \in A \ \& \ adr(a)=bz_{bz(a)}[val_P(\alpha)]\}$,

(3) $lf_P(\alpha) \ ::= \ \{a \mid a \in A \ \& \ a(val_P(\alpha)) \neq val_P(\alpha)\}$.

Es ist leicht zu sehen, daß $proc_P$ eine Äquivalenzrelation auf der Aktionenmenge A des Prozeßsystems $P=(A,D,w)$ ist. Sei P_i die i-te Äquivalenzklasse, d.h. $P_i = \{a \mid a \in A \ \& \ bz(a)=i\}$, dann kann P_i als der i-te Prozeß von P interpretiert werden. $bef_P(\alpha)$ ist die Menge derjenigen Aktionen, deren Adressen nach Ausführung der Folge α im Prozeßsystem P mit den aktuellen Befehlszählerständen übereinstimmen. In $lf_P(\alpha)$ sind alle diejenigen Aktionen enthalten, deren Ausführungsbedingung nach der Berechnung α in P erfüllt ist, d.h. die nach Ablauf von α lauffähig sind.
bef_P und lf_P besitzen drei wichtige Eigenschaften, die man mit den Analoga dieser Mengen in einem realen System intuitiv verbindet.

Satz 2.1.1.3/1: Sei $P=(A,D,w)$ ein aus n Prozessen bestehendes Prozeßsystem. Bezeichne $P_i ::= \{a \mid a \in A \ \& \ bz(a)=i\}$ den i-ten Prozeß, $i \in \{1, ..., n\}$. Dann besitzen bef_P und lf_P die folgenden Eigenschaften:

(1) Für jeden Ablauf $\alpha \in A^*$ gilt

$$lf_P(\alpha) \subset bef_P(\alpha).$$

(2) Ist $\alpha \in A^*$ und $i \in \{1, ..., n\}$, so gilt:

$$|(bef_P(\alpha) \cap P_i)| \leq 1.$$

(3) Sind $\alpha,\beta\in A^*$ und $i\in\{1, ..., n\}$, dann gilt:

$$[W(\beta)\cap P_i = \phi] \Rightarrow [P_i\cap bef_P(\alpha)=P_i\cap bef_P(\alpha\beta)].$$

Beweis: Sei $(bz_1, ..., bz_n, V_r)$ typisches Element von D, $a_i,a_j\in A$.

zu (1): *Annahme*: $a_i\in lf_P(\alpha)$. Dann muß nach Definition 2.1.1.2/1-(3) a_i die Form haben (mit $k=bz(a_i)$):

$$when\ \ bz_k=adr(a_i)\ \&\ B_{a_i}(V_r)\ \ do\ \ bz_k:=N_{a_i}(V_r);\ V_r:=F_{a_i}(V_r);\ \ od;$$

Nach Definition von lf_P muß $a_i(val_P(\alpha))\neq val_P(\alpha)$ gelten. D.h. die Ausführungsbedingung von a_i ist nach Ablauf von α in P erfüllt. Somit ist $bz_k[val_P(\alpha)]=adr(a_i)$ und daher $a_i\in bef_P(\alpha)$ nach Definition von bef_P.

zu (2): *Annahme*: $a_i,a_j\in bef_P(\alpha)\cap P_k$. Da a_i und a_j zum gleichen Prozeß gehören, muß $bz(a_i)=bz(a_j)=k$ gelten und wegen $a_i,a_j\in bef_P(\alpha)$ muß

$$adr(a_i)=bz_k[val_P(\alpha)]=adr(a_j)$$

erfüllt sein. Nach Definition 2.1.1.2/1-(4) sind in diesem Falle a_i und a_j identische Aktionen.

zu (3): *Es wird gezeigt*: Sind $k,l\in\{1, ..., n\}$, $k\neq l$, $a_i\in P_k$ und $a_j\in P_l$, dann gilt:

$$a_i\in bef_P(\alpha) \iff a_i\in bef_P(\alpha a_j).$$

Die Behauptung folgt dann unmittelbar mit Induktion über die Länge von β.
Annahme: $k,l\in\{1, ..., n\}$, $k\neq l$, $a_i\in P_k$ und $a_j\in P_l$. Aufgrund der Definition 2.1.1.2/1-(3) muß gelten: $bz_k[val_P(\alpha)]=bz_k[val_P(\alpha a_j)]$. Nach Definition von bef_P gilt $a_i\in bef_P(\alpha)$ genau dann, wenn $bz_k[val_P(\alpha)]=adr(a_i)$ und a_i ist in $bef_P(\alpha a_j)$ genau dann enthalten, wenn $bz_k[val_P(\alpha a_j)]=adr(a_i)$. Damit ist $a_i\in bef_P(\alpha) \iff a_i\in bef_P(\alpha a_j)$ gezeigt und der Rest der Behauptung folgt mit vollständiger Induktion. ■

Erläuterung von Satz 2.1.1.3/1: Informell besagt (1): Die Menge der lauffähigen Aktionen ist eine Teilmenge der Aktionen, deren Adressen mit den aktuellen Befehlszählerständen des Prozeßsystems übereinstimmen. (2) bedeutet: Die Aktionen eines Prozesses werden sequentiell ausgeführt. (3) kann interpretiert werden als: Aktionen können nur den Befehlszähler des Prozesses verändern, zu dem sie gehören.

In Definition 2.1.1.3/1 wurde der Begriff des Ablaufs eingeführt. Bei der Untersuchung der in einem Prozeßsystem ausführbaren Berechnungen kann man sich auf sogenannte aktive Abläufe (Berechnungen, Aktionenfolgen) beschränken.

Definition 2.1.1.3/4: Eine Berechnung $\alpha=\alpha[0]...\alpha[n]\in A^*$ wird als ein ***aktiver Ablauf*** des Prozeßsystems $P=(A,D,w)$ bezeichnet genau dann, wenn für jedes $i\in\{1, ..., n\}$ gilt:

$$\alpha[i] \in lf_P(\alpha[0]...\alpha[i-1]).$$

2.1.1.4. Veranschaulichung der Begriffe

Zur Veranschaulichung der eingeführten Begriffe dient das in Bild 2.1.1.4/1 angegebene Prozeßsystem $P=(A,D,w)$.
Die Aktionenmenge dieses Prozeßsystems is definiert als:

$$A ::= \{a_{1,1}, a_{1,2}, a_{1,3}, a_{1,4}, a_{2,1}, a_{2,2}, a_{2,3}, a_{2,4}\}$$

Die Zustandsmenge D ist gegeben durch:

$$D ::= \mathbf{N}_{[1,4]}\times\mathbf{N}_{[1,4]}\times(\mathbf{N}_0\times\mathbf{N}_0\times\mathbf{N}_0\times\mathbf{N}_0\times\mathbf{N}_0),$$

wobei $\mathbf{N}_{[1,4]} ::= \{1,2,3,4\}$ und $\mathbf{N}_0 ::= \{0,1,2, ...\}$.

$(bz_1, bz_2, (lv_{1,1}, lv_{2,1}, gv_1, gv_2, gv_3))$ ist typisches Element von D. Für das Anfangswort w gilt unter Zugrundelegung der in Bild 2.1.1.4/1 angegebenen Initialisierungswerte:

$$w ::= (1,1,(0,0,0,0,3)).$$

Mit der Notation $a_{i,j}$ zur Bezeichnung der j-ten Aktion des i-ten Prozesses von P gilt des weiteren:

$$bef_P(\lambda)=\{a_{1,1}, a_{2,1}\} \text{ und } lf_P(\lambda)=\{a_{1,1}, a_{2,1}\}.$$

Aktive Abläufe (Berechnungen) von P im Zustand $val_P(\lambda)$ sind die Folgen:

$$\{\lambda, a_{1,1}, a_{2,1}, a_{1,1}a_{1,2}, a_{1,1}a_{2,1}, a_{2,1}a_{1,1}, a_{1,1}a_{1,2}a_{1,3}, a_{1,1}a_{1,2}a_{2,1}, a_{1,1}a_{2,1}a_{1,2}, ...\}.$$

Der Ablauf $a_{2,1}a_{2,2}$ ist in P nicht aktiv, da $a_{2,2} \notin lf_P(a_{2,1})$.

Nach Ausführung der Berechnung $a_{1,1}a_{1,2}$ stellt sich der Systemzustand $val_P(a_{1,1}a_{1,2})=(3,1,(0,0,1,1,3))$ ein. In diesem Zustand gilt:

$$bef_P(a_{1,1}a_{1,2})=\{a_{1,3},a_{2,1}\} \text{ und } lf_P(a_{1,1}a_{1,2})=\{a_{1,3},a_{2,1}\}.$$

Aktive Berechnungen im Zustand $val_P(a_{1,1}a_{1,2})$ sind:

Anfangswerte: $gv_1 := 0;$ $gv_2 := 0;$ $gv_3 := 3;$
$bz_1 := 1;$ $lv_{1,1} := 0;$ $bz_2 := 1;$ $lv_{2,1} := 0;$

Prozeß P_1:

$a_{1,1}$: *when* $(bz_1 = adr(a_{1,1}))$ *do*
$bz_1 := (if\ (lv_{1,1} \geq gv_3)\ then\ adr(a_{1,4})\ else\ adr(a_{1,2}));$
od;

$a_{1,2}$: *when* $((bz_1 = adr(a_{1,2})) \& (gv_1 \leq 0))$ *do*
$gv_2 := gv_2 + 1;$
$gv_1 := gv_1 + 1;$
$bz_1 := adr(a_{1,3});$
od;

$a_{1,3}$: *when* $(bz_1 = adr(a_{1,3}))$ *do*
$lv_{1,1} := lv_{1,1} + 1;$
$bz_1 := adr(a_{1,1});$
od;

$a_{1,4}$: *when* $(bz_1 = adr(a_{1,4}))$ *do*
exit(P_1);
od;

Prozeß P_2:

$a_{2,1}$: *when* $(bz_2 = adr(a_{2,1}))$ *do*
$bz_2 := (if\ (lv_{2,1} \geq gv_3)\ then\ adr(a_{2,4})\ else\ adr(a_{2,2}));$
od;

$a_{2,2}$: *when* $((bz_2 = adr(a_{2,2})) \& (gv_1 \geq 1))$ *do*
$gv_2 := gv_2 * gv_2;$
$gv_1 := gv_1 - 1;$
$bz_2 := adr(a_{2,3});$
od;

$a_{2,3}$: *when* $(bz_2 = adr(a_{2,3}))$ *do*
$lv_{2,1} := lv_{2,1} + 1;$
$bz_2 := adr(a_{2,1});$
od;

$a_{2,4}$: *when* $(bz_2 = adr(a_{2,4}))$ *do*
exit(P_2);
od;

Bild 2.1.1.4/1: Beispiel-Prozeßsystem P

$\{\lambda, a_{1,3}, a_{2,1}, a_{1,3}a_{1,1}, a_{1,3}a_{2,1}, a_{2,1}a_{1,3}, a_{2,1}a_{2,2}, a_{1,3}a_{1,1}a_{2,1}, a_{1,3}a_{2,1}a_{1,1}, a_{1,3}a_{2,1}a_{2,2}, \ldots\}$.

Unter Beachtung der folgenden Zuordnungsvorschrift, kann das Beispiel-Prozeßsystem als eine mögliche, wenn auch etwas umständliche, Implementierung eines iterativen Verfahrens zur Berechnung der rekursiv definierten Funktion $f(x)$:

$$f(x) ::= \begin{cases} 0 & x=0 \\ & falls \\ (f(x-1)+1)^2 & x>0 \end{cases}$$

betrachtet werden.

Die zu treffende Zuordnung sieht wie folgt aus: x wird durch gv_3 repräsentiert und nach erfolgreichem Abschluß einer aktiven Berechnung nimmt die Variable gv_2 den Ergebniswert $f(gv_3[val_P(\lambda)])$ an. Dabei ist eine aktive Berechnungsfolge α genau dann als erfolgreich abgeschlossen anzusehen, wenn $lf_P(\alpha) = \phi$.

Die restlichen Variablen des Beispiel-Prozeßsystems, und dementsprechend auch die Aktionen, in welchen sie vorkommen, haben die folgende Funktion: Den Variablen $lv_{1,1}$ und $lv_{2,1}$ fällt die Aufgabe zu die Zahl der Ausführungen von Aktion $a_{1,1}$ bzw. $a_{2,1}$ innerhalb einer Berechnungsfolge festzuhalten. Nach einer durch den Wert der Variablen gv_3 begrenzten Anzahl von Aktivierungen der Aktionen $a_{1,1}$ und $a_{2,1}$ wird jeder aktive Ablauf durch Ausführung der Aktionen $a_{1,4}$ und $a_{2,4}$ bzgl. des Prozesses P_1 bzw P_2 ordnungsgemäß beendet.

Bei der Ermittlung des Funktionswertes $f(gv_3[val_P(\lambda)])$ übernimmt P_1 das Addieren und P_2 das Multiplizieren. Für die alternierende Ausführung dieser Operationen sorgt die Variable gv_1, die in der Ausführungsbedingung der Aktionen $a_{1,2}$ und $a_{2,2}$ auf kleiner oder gleich 0 bzw größer oder gleich 1 abgefragt wird, was dazu führt, daß zu jedem Zeitpunkt nur eine dieser beiden Aktionen ausführbar sein kann.

In der vorliegenden Beispielsituation führt im Zustand $val_P(\lambda)$ der aktive Ablauf $(a_{1,1}a_{1,2}a_{1,3}a_{2,1}a_{2,2}a_{2,3})^3a_{1,1}a_{1,4}a_{2,1}a_{2,4}$ zu einem erfolgreichen Berechnungsabschluß und zu einem Endzustand, in dem gv_2 den Wert 25 besitzt. Dies entspricht dem Wert der Funktion $f(x)$ an der Stelle $x=gv_3[val_P(\lambda)]=3$

Eine mögliche Abbildung des Beispiel-Prozeßsystems auf den Arbeitsspeicher einer digitalen Datenverarbeitungsanlage zeigt Bild 2.1.1.4/2. Eine entsprechende Verallgemeinerung auf n Prozesse (n Code- und m Daten-Segmente) ist offensichtlich.

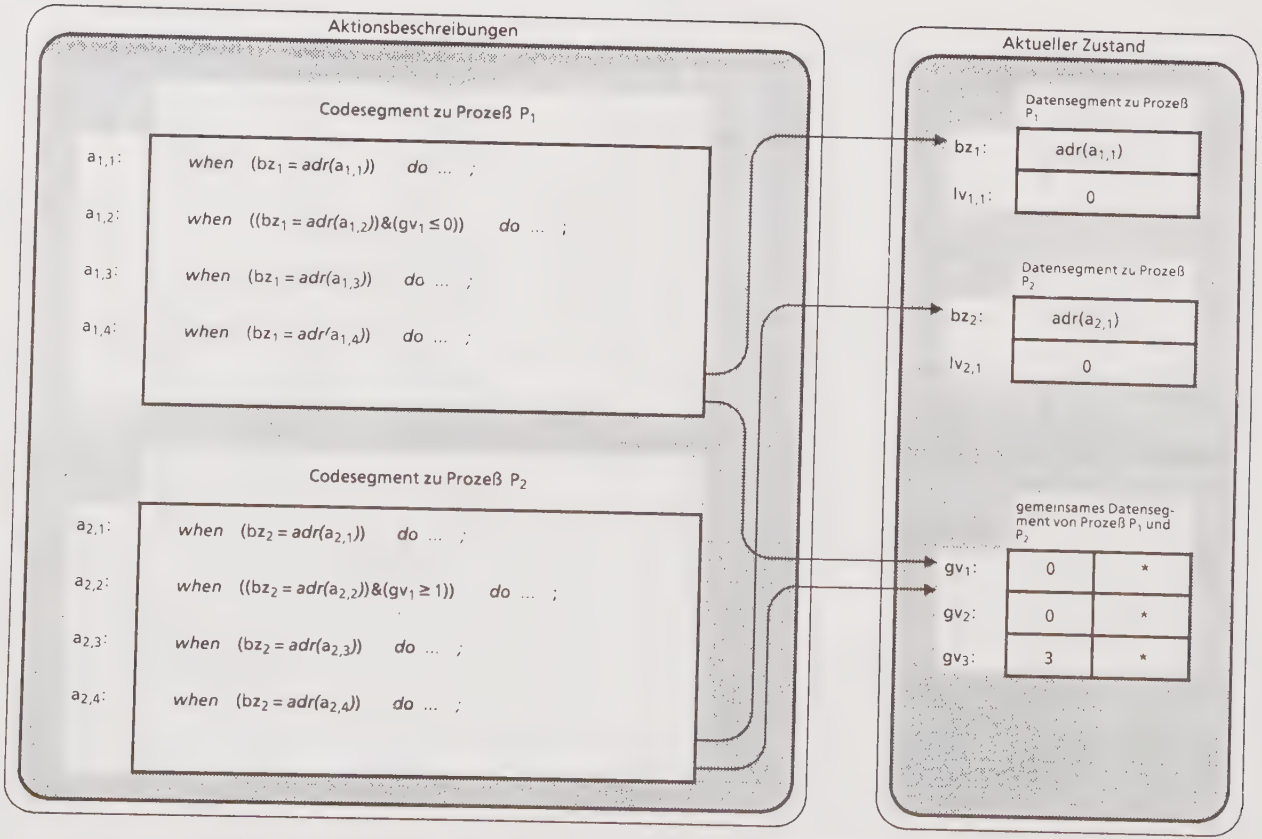


Bild 2.1.1.4/2: Abbildung eines Prozeßsystems auf einen Arbeitsspeicher am Beispiel von P

Der in Bild 2.1.1.4/2 gewählten Darstellung liegt die implizite Annahme zugrunde, daß die in den Aktionsbeschreibungen enthaltenen Bezugnahmen auf Variable über virtuelle Adressen erfolgen. Der skizzierte Prozeßsystem-Zustand ist $val_P(\lambda)$.

Ausführliche Erläuterungen sowohl zur Abbildung von Prozeßsystemen in ausführbarer Form auf Arbeitsspeicher digitaler Rechenanlagen als auch zum Thema virtuelle Adressierung findet man in [Hoß84].

2.1.2. Aktionentypen

Anhand der Art der Bearbeitung und des Geltungsbereiches können innerhalb der Aktionenmenge A eines Prozeßsystems $P=(A,D,w)$ vier verschiedene Typen von Aktionen unterschieden werden:

(1) *lesende und schreibende Aktionen*,
(2) *lokale und globale Aktionen*,
(3) *nachrichtenübertragende* und
(4) *synchronisierende Aktionen*.

Die Unterscheidung dieser Aktionentypen wird sich für die im Rahmen dieser Arbeit durchzuführenden Untersuchungen als sehr nützlich erweisen. Das Prozeßsystem-Modell von R.J. Lipton, [Lip74], ermöglicht eine einfache Präzisierung der in diesem Zusammenhang neu einzuführenden Begriffe.

2.1.2.1. Lesende und schreibende Aktionen

Um, bezogen auf jeden beliebigen Zustand eines Prozeßsystems, Aussagen über das Ausmaß einer im Fehlerfall notwendigen Rücksetzausbreitung machen zu können, ist die Berücksichtigung der im Laufe einer Berechnung entstandenen *Lese- und Schreib-Referenzen von Aktionen zu Variablen* wesentlich (vgl. die einführenden Beispiele des 1. Kapitels). Dies führt zu den folgenden Vereinbarungen, auf die später zurückgegriffen wird.

Definition 2.1.2.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem.

(1) Eine Aktion $a \in A$ *greift auf* eine Variable v *lesend zu* genau dann, wenn v im Bedingungsteil von a, in der Ausführungsbedingung einer if-Abfrage innerhalb von a oder auf der 'rechten Seite' einer Wertzuweisung innerhalb von a vorkommt.
Etwas abstrakter formuliert greift $a \in A$ auf v lesend zu genau dann, wenn die Ausführbarkeit von a und/oder das Ergebnis von a vom Wert der Variablen v vor Ausführung von a abhängt.

Kurzschreibweise $L(a)$=**true** :<=> a ist Leseaktion, d.h. a greift auf eine Variable v lesend zu.

(2) Eine Aktion $a \in A$ *greift auf* eine Variable v *schreibend zu* genau dann, wenn v auf der 'linken Seite' einer Wertzuweisung innerhalb von a vorkommt.

Kurzschreibweise: $S(a)$=**true** :<=> a ist Schreibaktion, d.h. a greift auf eine

Variable v schreibend zu

Da der Befehlszähler $bz_{bz(a)}$ nach Definition 2.1.1.2/1-(3) innerhalb jeder Aktion a eines Prozeßsystems sowohl auf der linken Seite einer Wertzuweisung als auch in der Ausführungsbedingung von a vorkommt, folgt mit der Definition 2.1.2.2/1, daß jede Aktion $a \in A$ eines Prozeßsystems $P=(A,D,w)$ sowohl eine Leseaktion als auch eine Schreibaktion ist. Für das weitere ist deshalb vor allem die Teilmenge der *globalen Lese- und Schreibaktionen* eines Prozeßsystems von Bedeutung, zu deren präzisen Charakterisierung die folgende Definition nützlich ist.

Definition 2.1.2.1/2: Sei $P=(A,D,w)$ ein n Prozesse umfassendes Prozeßsystem.

(1) Mit LV_a (SV_a) wird die Teilmenge derjenigen Variablen von V_a (vgl. Definition 2.1.2.2/1) bezeichnet, auf welche die Aktion $a \in A$ lesend (schreibend) zugreift.

(2) LV_{P_i} (SV_{P_i}) steht entsprechend für die Menge der Variablen, auf die Aktionen des Prozesses P_i von P lesend (schreibend) zugreifen:

$$LV_{P_i} ::= \bigcup_{a \in A \cap P_i} LV_a,\ SV_{P_i} ::= \bigcup_{a \in A \cap P_i} SV_a \text{ und } P_i ::= \{a \mid a \in A \ \&\ bz(a)=i\}.$$

(3) LV_P (SV_P) ist schließlich definiert als die Menge aller Variablen, auf welche Prozesse von P lesend (schreibend) zugreifen, d.h.:

$$LV_P ::= \bigcup_{i=1}^{n} LV_{P_i} \text{ und } SV_P ::= \bigcup_{i=1}^{n} SV_{P_i}.$$

Satz 2.1.2.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem und $a \in A$. Zwischen den Variablenmengen LV_x, SV_x und V_x, $x \in \{a, P_i, P\}$, von P gilt die folgende Beziehung:

$$LV_x \cup SV_x = V_x.$$

Beweis: $LV_a \cup SV_a \subseteq V_a$ folgt unmittelbar mit der Definition von LV_a und SV_a. $V_a \subseteq LV_a \cup SV_a$ folgt aus der Tatsache, daß auf jede in einer Aktion referierte Variable lesend und/oder schreibend zugegriffen wird. $LV_{P_i} \cup SV_{P_i} = V_{P_i}$ und $LV_P \cup SV_P = V_P$ folgen unmittelbar aufgrund der nachgewiesenen Beziehung $LV_a \cup SV_a = V_a$ und der Definition von LV_{P_i}, SV_{P_i} sowie V_{P_i} bzw. LV_P, SV_P und V_P auf der Basis von LV_a, SV_a und V_a (zur Bezeichnung V_x, $x \in \{a, P_i, P\}$, vgl. Definition 2.1.2.2/1). ■

Definition 2.1.2.1/3: Sei $P=(A,D,w)$ ein Prozeßsystem

(1) Eine Aktion $\alpha_i \in A$ wird als *globale Leseaktion* bezeichnet genau dann, wenn gilt:

$$\exists \alpha_j \in A \; [\neg proc(\alpha_i, \alpha_j) \; \& \; LV_{\alpha_i} \cap V_{\alpha_j} \neq \phi].$$

Kurzschreibweise: $gL(\alpha_i)$=**true** ::<=> α_i ist globale Leseaktion.

(2) Eine Aktion $\alpha_i \in A$ wird als *globale Schreibaktion* bezeichnet genau dann, wenn gilt:

$$\exists \alpha_j \in A \; [\neg proc(\alpha_i, \alpha_j) \; \& \; SV_{\alpha_i} \cap V_{\alpha_j} \neq \phi].$$

Kurzschreibweise: $gS(\alpha_i)$=**true** ::<=> α_i ist globale Schreibaktion.

Beispiel 2.1.2.1/1: Als globale Leseaktionen können z.B. die Aktionen $\alpha_{1,1}$, $\alpha_{1,2}$, $\alpha_{2,1}$ und $\alpha_{2,2}$ des in Bild 2.1.1.4/1 beschriebenen Beispiel-Prozeßsystems P bezeichnet werden. Hierbei sind $\alpha_{1,2}$ und $\alpha_{2,2}$ gleichzeitig auch globale Schreibaktionen. gv_1 und gv_2 sind sowohl globale Lese- als auch globale Schreibvariable. Im Gegensatz zu gv_3, die lediglich als globale Lesevariable fungiert.

2.1.2.2. Lokale und globale Aktionen

In dem vorangehend skizzierten Beispiel 2.1.1.4/1 wurde bei der Benennung der darin vorkommenden Variablen bewußt darauf geachtet, daß *prozeßlokale (kurz: lokale) Variablen*, mit Ausnahme der Befehlszähler, durch Voranstellen des Buchstabens '*l*' im Namen deutlich von den *prozeßgemeinsamen (globalen) Variablen*, deren Namen wurden durch ein '*g*' eingeleitet, unterschieden werden konnten. Die auf diese Weise zum Ausdruck gebrachte Unterscheidung ist in bezug auf die zu behandelnden Fragestellungen aus folgendem Grunde, unabhängig von Problemen der Abbildung eines Prozeßsystems auf einen realen Arbeitsspeicher und der Zusammenfassung globaler Variablen in von Prozessen gemeinsam benutzte Datensegmente (vgl. Bild 2.1.1.4/2), wesentlich:

Eine der Voraussetzungen für eine *fortschreitende Rücksetzausbreitung (den Dominoeffekt)* im Rahmen einer Fehlerbehandlung ist das Bestehen von Abhängigkeiten zwischen den Prozessen eines Prozeßsystems. Da die Bearbeitung, d.h. das Lesen, Schreiben und das Ändern, jeder lokalen Variablen jeweils ausschließlich durch einen einzigen Prozeß P_i eines Prozeßsystems P erfolgt, können Abhängigkeiten zwischen zwei Prozessen P_i und P_j, $i \neq j$, eines Prozeßsystems nicht unmittelbar über lokale Variablen entstehen. Ebenso können sich Fehler innerhalb eines Prozeßsystems P nicht unmittelbar über lokale Variablen von einem Prozeß P_i zu einem anderen Prozeß P_j, $i \neq j$, fortpflanzen. Beides ist nur über globale, also prozeßgemeinsame Variablen möglich. Dies bestätigen auch die Beispiele des ersten Kapitels.
Das bedeutet: Sollen Aussagen über Möglichkeiten zur Begrenzung der Rücksetzausbreitung gemacht werden, ist eine Klassifizierung der Variablen in

lokale und globale sowie eine entsprechende Präzisierung dieser Begriffe erforderlich.

Bei der Definition der beiden Begriffe lokale Variable und globale Variable wird auf die folgende Vereinbarung zurückgegriffen.

Definition 2.1.2.2/1: Sei $P=(A,D,w)$ ein n Prozesse umfassendes Prozeßsystem

(1) Mit V_α wird die Menge der Variablen bezeichnet, auf welche die Aktion α, $\alpha \in A$, zugreift (inklusive des Befehlszählers $bz_{bz(\alpha)}$).

(2) V_{P_i} steht stellvertretend für die Menge der Variablen, auf die Aktionen des Prozesses P_i von P zugreifen, d.h.:

$$V_{P_i} ::= \bigcup_{\alpha \in A \cap P_i} V_\alpha ,$$

wobei $P_i ::= \{\alpha \mid \alpha \in A \ \& \ bz(\alpha)=i\}$.

(3) Schließlich ist V_P definiert als die Menge aller Variablen, auf welche Prozesse des Prozeßsystems P zugreifen:

$$V_P ::= \bigcup_{i=1}^{n} V_{P_i}.$$

Die Begriffe lokale und globale Variable lassen sich mit Hilfe der in Definition 2.1.2.1/1 eingeführten Bezeichnungsweisen wie folgt präzisieren.

Definition 2.1.2.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem.

(1) Eine Variable $v \in V_P$ wird als *lokale Variable* bezeichnet genau dann, wenn gilt:

$$\forall \alpha_i,\alpha_j \in A \ [\neg proc_P(\alpha_i,\alpha_j) \Rightarrow v \notin V_{\alpha_i} \cap V_{\alpha_j}].$$

Kurzschreibweise: $l(v)=\mathbf{true}$ ::<=> v ist lokale Variable.

(2) Eine Variable $v \in V_P$ wird als *globale Variable* bezeichnet genau dann, wenn gilt:

$$\exists \alpha_i,\alpha_j \in A \ [\neg proc_P(\alpha_i,\alpha_j) \ \& \ v \in V_{\alpha_i} \cap V_{\alpha_j}].$$

Kurzschreibweise: $g(v)=\mathbf{true}$ ::<=> v ist globale Variable

(3) $yV_x ::= y(V_x)=\{v \mid v \in V_x \ \& \ y(v)=\mathbf{true}\}$, wobei:

$$y \in \{l,g\},\ x \in \{\alpha,P_i,P\},\ \alpha \in A \text{ und } P_i ::= \{\alpha \mid \alpha \in A \ \& \ bz(\alpha)=i\}.$$

Entsprechend der Unterteilung in lokale und globale Variablen läßt sich auch die Aktionenmenge A eines Prozeßsystems P unterteilen.

Definition 2.1.2.2/3: Sei $P=(A,D,w)$ ein Prozeßsystem.

(1) Eine Aktion $a_i \in A$ wird als *lokale Aktion* bezeichnet genau dann, wenn gilt:

$$\forall a_j \in A\ [\neg proc_P(a_i,a_j) \Rightarrow V_{a_i} \cap V_{a_j} = \phi].$$

Kurzschreibweise: $l(a_i)=\textbf{true}$ ::<=> a_i ist lokale Aktion.

(2) Eine Aktion $a_i \in A$ wird als *globale Aktion* bezeichnet genau dann, wenn gilt:

$$\exists a_j \in A\ [\neg proc_P(a_i,a_j)\ \&\ V_{a_i} \cap V_{a_j} \neq \phi].$$

Kurzschreibweise: $g(a_i)=\textbf{true}$ ::<=> a_i ist globale Aktion.

(3) yA_x ::= $y(A_x)=\{a \mid a \in A_x\ \&\ y(a)=\textbf{true}\}$, wobei:

$$y \in \{l,g\},\ x \in \{P_i,P\},\ P_i ::= \{a \mid a \in A\ \&\ bz(a)=i\},\ A_P ::= A \text{ und } A_{P_i} ::= P_i.$$

Beispiel 2.1.2.2/1: Unter Zugrundelegung der vorstehenden Definitionen sind die Aktionen $a_{1,3}$, $a_{1,4}$, $a_{2,3}$ und $a_{2,4}$ des im Bild 2.1.1.4/1 beschriebenen Beispiel-Prozeßsystems P lokale Aktionen; $a_{1,1}$, $a_{1,2}$, $a_{2,1}$ sowie $a_{2,2}$ sind globale Aktionen.
Wie aus der Bezeichnungsweise der Variablen von P bereits hervorgeht sind gv_1, gv_2 und gv_3 globale Variablen, die den Prozessen P_1 und P_2 gemeinsam sind; bz_1, $lv_{1,1}$ und bz_2, $lv_{2,1}$ sind hingegen lokale Variablen, die nur von P_1 bzw. P_2 referiert werden.

Der folgende Satz zeigt, daß die Unterscheidung in lokale und globale Variablen (Aktionen) eine vollständige Partition auf den Mengen V_a, V_{P_i} und V_P (bzw. A_{P_i} und A_P) bildet.

Satz 2.1.2.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem und $a \in A$. Bzgl. der Variablen- und Aktionenmengen yV_x und yA_x, $x \in \{a, P_i, P\}$, $y \in \{l,g\}$, $z \in \{P_i, P\}$, gilt:

(1) $lV_x \cap gV_x = \phi$,
(2) $lV_x \cup gV_x = V_x$,
(3) $lA_x \cap gA_x = \phi$,
(4) $lA_x \cup gA_x = A_x$.

Beweis: Folgt unmittelbar mit den Definitionen der verschiedenen Variablen- und Aktionenmengen yV_x bzw. yA_x. ■

2.1.2.3. Nachrichtenübertragende Aktionen

Jeder Kommunikationsfluß zwischen Prozessen kann zwar generell auf der Basis von Lese- und Schreibaktionen beschrieben werden. Häufig basiert er jedoch auf speziellen höherwertigen *nachrichtenübertragenden Operationen*, die zwar mit Hilfe von Lese- und Schreibaktionen realisiert werden, von diesen aber zu abstrahieren gestatten, und auf diese Weise die Modellierung von Kommunikationsbeziehungen zwischen Prozessen vereinfachen. Diese Beobachtung führt zur Definition eines *Nachrichtensystems*.

Definition 2.1.2.3/1: Ein Prozeßsystem $P=(A,D,w)$ wird als Γ-*Nachrichtensystem* bezeichnet, wenn es folgende Eigenschaften besitzt (Γ steht stellvertretend für einen konkreten Systemnamen):

(1) Die Zustandsmenge D hat die Gestalt

$$D=D_1\times\ldots\times D_n\times(\mathbb{N}_0{}^2\times D_N{}^{k_1})\times\ldots\times(\mathbb{N}_0{}^2\times D_N{}^{k_m})\times D_N{}^p\times D_{V_r};$$

typisches Element ist:

$$(bz_1,\ldots,bz_n,(f_1,b_1,np_1),\ldots,(f_m,b_m,np_m),nv_1,\ldots,nv_p,V_r).$$

Die Variablen $np_1,\ldots np_m$ werden als Nachrichtenpuffer bezeichnet; die Variablen b_i zeigen den Füllungsgrad des Puffers np_i an und die Variablen f_i die Zahl der freien Plätze innerhalb von np_i, $i\in\{1,\ldots,m\}$. Die Kapazität k_i des i-ten Nachrichtenpuffers errechnet sich zu: $f_i[w]+b_i[w]$.
Die Variablen $nv_1,\ldots,nv_p$ sind Variablen vom Typ Nachricht. Unter der Bezeichnung V_r sind die restlichen Variablen von P zusammengefaßt. $\mathbb{N}_0$, D_N und D_{V_r} sind die entsprechenden Definitionsbereiche.

(2) Alle Aktionen $a\in A$ haben entweder die Form:

$$\begin{aligned}a:\ &when\ \ bz_{bz(a)}=adr(a)\ \ do\ \ bz_{bz(a)}:=N_a(nv_1,\ldots,nv_p,V_r);\\ &\qquad\qquad\qquad\qquad\quad V_r:=F_a(nv_1,\ldots,nv_p,V_r);\\ &od;\end{aligned}$$

und heißen *nichtnachrichtenübertragend* oder eine der beiden Formen:

$$\begin{aligned}a:\ &when\ \ bz_{bz(a)}=adr(a)\ \&\ f_i>0\ \ do\\ &\qquad\qquad\qquad\qquad bz_{bz(a)}=adr(a)';\\ &\qquad\qquad\qquad\qquad f_i:=f_i-1;\\ &\qquad\qquad\qquad\qquad b_i:=b_i+1;\\ &\qquad\qquad\qquad\qquad \Theta_{ein}(np_i,nv_j);\\ &\qquad\qquad od;\end{aligned}$$

$$\begin{aligned}a:\ &when\ \ bz_{bz(a)}=adr(a)\ \&\ b_i>0\ \ do\\ &\qquad\qquad\qquad\qquad bz_{bz(a)}=adr(a)';\\ &\qquad\qquad\qquad\qquad b_i:=b_i-1;\\ &\qquad\qquad\qquad\qquad f_i:=f_i+1;\end{aligned}$$

$\Theta_{aus}(np_i, nv_j)$;
od;

und heißen *nachrichtenübertragend*.

Kurzschreibweise: $n(a)$=**true** ::<=> a ist nachrichtenübertragende Aktion.

Die Operation $\Theta_{ein}(np_i, nv_j)$ fügt die Nachricht nv_j in den Puffer np_i ein. $\Theta_{aus}(np_i, nv_j)$ weist als Ergebnis ein Element der in np_i stehenden Nachrichten-Sequenz der Variablen nv_j zu und entfernt dieses anschließend aus np_i.
Mit $adr(a)'$ wird die unmittelbar auf $adr(a)$ im Prozeß $P_{bz(a)}$ folgende Aktionsadresse bezeichnet. Im Unterschied dazu, kann mit der Funktion $N_a(...)$ dem Befehlszähler des Prozesses $P_{bz(a)}$ jede beliebige Aktionsadresse, die im Adreßraum des Prozesses liegt, als Nachfolgeradresse von $adr(a)$ zugewiesen werden.

Definition 2.1.2.3/2: Sei $P=(A,D,w)$ ein Γ-Nachrichtensystem. Eine Variable $v \in V_P$ wird als *Nachrichtenvariable* bezeichnet genau dann, wenn ihr Definitionsbereich vom Typ D_N ist.

Kurzschreibweise: $n(v)$=**true** ::<=> v ist Nachrichtenvariable.

Die abstrakte Definition eines Γ-Nachrichtensystems erlaubt durch Ersetzung der Metavariablen Θ_{ein}, Θ_{aus} und Γ unterschiedliche Nachrichtensysteme, die sich durch die Strategien, nach welchen Nachrichten in einen Nachrichtenpuffer eingefügt oder aus einem Nachrichtenpuffer entnommen werden, unterscheiden, zu beschreiben. Drei dieser Strategien sind in Tabelle 2.1.2.3/1 skizziert.

Notationsvereinbarung: Ist a eine nachrichtenübertragende Aktion, die eine Nachricht nv_j in einen Nachrichtenpuffer np_i einfügt, so wird im folgenden a durch

$$a: send_\Gamma(np_i, nv_j);$$

repräsentiert; entsprechend wird eine nachrichtenübertragende Aktion, die eine Nachricht aus einem Nachrichtenpuffer np_i entnimmt und diese der Variablen nv_j zuweist, abkürzend als

$$a: receive_\Gamma(np_i, nv_j);$$

dargestellt.

Beispiel 2.1.2.3/1: Ein Γ-Nachrichtensystem Q, wobei in diesem Falle Γ∈{FIFO, LIFO, RANDOM} beliebig gewählt werden könnte, da die Kapazität der verwendeten Puffer genau 1 ist, zeigt Bild 2.1.2.3/1.

Systemname	Θ_{ein}	Θ_{aus}
FIFO	$np_i[\,|np_i| + 1] := nv_j;$	$nv_j := np_i[1];$ $np_i[k-1] := np_i[k];$ $np_i[\,|np_i|\,] := \lambda;$ $\forall k: 2 \leq k \leq |np_i|$
LIFO	$np_i[\,|np_i| + 1] := nv_j;$	$nv_j := np_i[\,|np_i|\,];$ $np_i[\,|np_i|\,] := \lambda;$
RANDOM	$np_i[\,|np_i| + 1] := nv_j;$	$nv_j := np_i[k];$ $np_i[k + l - 1] := np_i[k + l];$ $1 \leq k \leq |np_i|$ $\forall l: 1 \leq l \leq |np_i| - k$

Tabelle 2.1.2.3/1: Beispiele für zu unterscheidende Γ-Nachrichtensysteme

Anfangswerte: $nv := N$; $np_1 := N$; $np_2 := \emptyset$; $gv_2 := 0$; $gv_3 := 3$;
$bz_1 := 1$; $lv_{1,1} := 0$; $bz_2 := 1$; $lv_{2,1} := 0$;

Prozeß Q_1:

$a_{1,1}$: *when* $(bz_1 = adr(a_{1,1}))$ *do*
$bz_1 := (if\ (lv_{1,1} \geq gv_3)\ then\ adr(a_{1,4})\ else\ adr(a_{1,2}));$
od;

$a_{1,(2,1)}$: $receive_{FIFO}(np_1, nv);$
$a_{1,(2,2)}$: *when* $(bz_1 = adr(a_{1,(2,2)}))$ *do*
$gv_2 := gv_2 + 1;$
$bz_1 := adr(a_{1,(2,3)});$
od;
$a_{1,(2,3)}$: $send_{FIFO}(np_2, nv);$

$a_{1,3}$: *when* $(bz_1 = adr(a_{1,3}))$ *do*
$lv_{1,1} := lv_{1,1} + 1;$
$bz_1 := adr(a_{1,1});$
od;

$a_{1,4}$: *when* $(bz_1 = adr(a_{1,4}))$ *do*
exit(P_1);
od;

Prozeß Q_2:

$a_{2,1}$: *when* $(bz_2 = adr(a_{2,1}))$ *do*
$bz_2 := (if\ (lv_{2,1} \geq gv_3)\ then\ adr(a_{2,4})\ else\ adr(a_{2,2}));$
od;

$a_{2,(2,1)}$: $receive_{FIFO}(np_2, nv);$
$a_{2,(2,2)}$: *when* $(bz_2 = adr(a_{2,(2,2)}))$ *do*
$gv_2 := gv_2 * gv_2;$
$bz_2 := adr(a_{2,(2,3)});$
od;
$a_{2,(2,3)}$: $send_{FIFO}(np_1, nv);$

$a_{2,3}$: *when* $(bz_2 = adr(a_{2,3}))$ *do*
$lv_{2,1} := lv_{2,1} + 1;$
$bz_2 := adr(a_{2,1});$
od;

$a_{2,4}$: *when* $(bz_2 = adr(a_{2,4}))$ *do*
exit(P_2);
od;

Bild 2.1.2.3/1: Beispiel-FIFO-Nachrichtensystem Q

Man überzeugt sich leicht, daß Q im Falle der folgenden Abbildung von Aktionen von Q auf diejenigen des in Bild 2.1.1.4/1 festgehaltenen Beispiel-Prozeßsystems P, genau die gleichen Berechnungen wie P zuläßt und des weiteren zum selben Ergebnis führt. Die zu treffende Zuordnung sieht wie folgt aus: $(a_{i,j}\ in\ Q) \rightarrow (a_{i,j}\ in\ P)$ im Falle $i \in \{1,2\}$ und $j \in \{1,3,4\}$; $(a_{1,(2,2)}\ in\ Q) \rightarrow (a_{1,2}\ in\ P)$, $(a_{2,(2,2)}\ in\ Q) \rightarrow (a_{2,2}\ in\ P)$ und $(a_{i,(2,k)}\ in\ Q) \rightarrow \lambda$ für $i \in \{1,2\}$ und $k \in \{1,3\}$.

2.1.2.4. Synchronisierende Aktionen

Ein weiteres wichtiges Unterscheidungsmerkmal, auf das in dieser Arbeit zurückgegriffen wird, stellt schließlich die Unterteilung der Variablen- und Aktionenmenge eines Prozeßsystems in *synchronisierende* und *nichtsynchronisierende Variablen* bzw. *Aktionen* dar.

Anschaulich gesprochen kann jede Variable, die in einer Ausführungsbedingung einer Aktion a eines Prozeßsystems P vorkommt, als Synchronisationsvariable bezeichnet werden. Alle anderen Variablen von P sind als nichtsynchronisierend einzustufen, da sie die Lauffähigkeit einer Aktion nicht beeinflussen können.

Eine ausgezeichnete Klasse von Synchronisationsvariablen sind die *Befehlszähler*. Wie bereits erwähnt, sorgt der Befehlszähler eines Prozesses dafür, daß zu einem Zeitpunkt nur genau eine Aktion $a_{i,j} \in P_i$ aktiv werden kann. Über ihn wird auch die Ausführungsreihenfolge von Aktionen eines Prozesses gesteuert, d.h. über ihn erfolgt eine implizite Synchronisation, die als *Intraprozeß-Synchronisation* bezeichnet werden kann.
Befehlszähler gehören zu den lokalen Variablen eines Prozeßsystems.

Da in der Ausführungsbedingung jeder Aktion $a \in A$ eines Prozeßsystems $P=(A,D,w)$ ein Befehlszähler implizit oder explizit vorkommt, ist es naheliegend jede Aktion a von P als eine (prozeßlokal) synchronisierende Aktion zu bezeichnen. Wird jedoch im folgenden der Begriff synchronisierende Aktion ohne den Zusatz 'prozeßlokal' verwendet, so bezieht er sich auf eine Klasse von Aktionen, welche die Modellierung der Koordination (quasi-) paralleler Abläufe innerhalb eines Prozeßsystems gestatten. Diese Form der sogenannten *Interprozeß-Synchronisation* ermöglichen *Variablen vom Typ Semaphor*. Variablen dieses Typs wurden erstmals von E.W. Dijkstra in die Literatur eingeführt [Dij68]. Sie bilden eine Teilmenge der globalen Variablen eines Prozeßsystems und werden in der Regel nur von speziellen Aktionstypen, im folgenden werden diese als *synchronisierende Aktionen* bezeichnet, verändert.
Das Konzept der Semaphorvariablen ist eng verknüpft mit dem Begriff

Synchronisationssystem.

Definition 2.1.2.4/1: Ein Prozeßsystem $P=(A,D,w)$ wird als Γ-*Synchronisationssystem* bezeichnet, wenn es die folgenden Eigenschaften besitzt (Γ steht stellvertretend für einen konkreten Systemnamen):

(1) Die Zustandsmenge D hat die Gestalt $D_1\times\ldots\times D_n\times\mathbf{Z}^m\times D_{V_r}$. Typisches Element ist $(bz_1,\ldots,bz_n,sv_1,\ldots,sv_m,V_r)$. Die Variablen $sv_1,\ldots,sv_m$ heißen *Semaphorvariablen*; ihr Wertebereich ist die Menge der ganzen Zahlen $\mathbf{Z}$. Die restlichen Variablen von P sind unter der Bezeichnung V_r zusammengefaßt; die zugehörigen Definitionsbereiche dieser Variablen sind in D_{V_r} festgehalten.

(2) Alle Aktionen $a\in A$ haben entweder die Form:

$$a\text{: }\textit{when}\ \ bz_{bz(a)}=adr(a)\ \ \textit{do}\ \ bz_{bz(a)}:=N_a(V_r);\ V_r:=F_a(V_r);\ \textit{od};$$

und heißen *nichtsynchronisierend* oder die Form:

$$a\text{: }\textit{when}\ \ bz_{bz(a)}=adr(a)\ \&\ \Phi_a(sv_1,\ldots,sv_m)\ \ \textit{do}\ \ bz_{bz(a)}=adr(a)';\ \Theta_a(sv_1,\ldots,sv_m);\ \textit{od};$$

und heißen *synchronisierend*, wobei $\Phi_a(sv_1,\ldots,sv_m)$ ein Prädikat über den Semaphoren von P und $\Theta_a(sv_1,\ldots,sv_m)$ eine Menge von Zuweisungen, die nur von Semaphoren abhängen und nur Semaphore ändern, ist; mit $adr(a)'$ wird die unmittelbar auf $adr(a)$ im Prozeß $P_{bz(a)}$ folgende Adresse bezeichnet.

Kurzschreibweise: $s(a)=\textbf{true}$ ::<=> a ist synchronisierende Aktion.

Zur Motivation des Synchronisationssystem-Begriffs siehe auch [Bat82] und [Hof84].

Definition 2.1.2.4/2: Sei $P=(A,D,w)$ ein Γ-Synchronisationssystem. Eine Variable $v\in V_P$ wird als *Semaphorvariable* bezeichnet genau dann, wenn:

$$\exists a\in A\ [s(a)=\textbf{true}\ \&\ v\in\Phi_a\cup\Theta_a].$$

Kurzschreibweise: $s(v)=\textbf{true}$ ::<=> v ist Semaphorvariable.

Semaphorvariablen können als Teilmenge der (*Ausführungs-*)*Bedingungsvariablen* (auch *Synchronisationsvariablen* genannt) eines Prozeßsystems angesehen werden. Diese lassen sich wie folgt charakterisieren.

Definition 2.1.2.4/3: Sei $P=(A,D,w)$ ein Prozeßsystem. Eine Variable $v \in V_P$ wird als *(Ausführungs-)Bedingungsvariable* (oder *Synchronisationsvariable*) bezeichnet genau dann, wenn gilt:

$$\exists a \in A\ [v \in B_a \parallel v = bz_{bz(a)}].$$

Kurzschreibweise $b(v)$=**true** :<=> v ist *(Ausführungs-)Bedingungsvariable*.

Die abstrakte Definition eines Γ-Synchronisationssystems erlaubt durch Ersetzung der Metavariablen Φ, Θ und Γ die unterschiedlichsten sowohl für die Realisierung von Betriebssystemen, Tabelle 2.1.2.4/1, als auch für theoretische Untersuchungen, siehe Tabelle 2.1.2.4/2, wichtigen Synchronisationssysteme zu beschreiben, vgl. [Hof84]. Eine ausführliche Beschreibung von insgesamt mehr als 40 verschiedenen Synchronisationssystemen findet man auch in [Bat82].
Das folgende Beispiel veranschaulicht die in diesem Abschnitt eingeführten Begriffe.

Beispiel 2.1.2.4/1: Die Variable gv_1 sowie die Befehlszähler bz_1 und bz_2 des im Bild 2.1.2.4/1 dargestellten Beispiel-Prozeßsystems P sind Bedingungsvariablen im Sinne von Definition 2.1.2.4/3.
Bild 2.1.2.4/1 zeigt wie P beispielsweise abgeändert werden kann, um der Definition eines *PV*-Synchronisationssystems zu genügen.
Die Rolle der Variablen gv_1 wird in dem in Bild 2.1.2.4/1 skizzierten Synchronisationssystem Q von den Semaphorvariablen sv_1 und sv_2 übernommen. Die Aktionen $\alpha_{1,2}$ und $\alpha_{2,2}$ von P werden in Q durch die Aktionen $\alpha_{1,(2,1)}$, $\alpha_{1,(2,2)}$, $\alpha_{1,(2,3)}$ bzw. $\alpha_{2,(2,1)}$, $\alpha_{2,(2,2)}$, $\alpha_{2,(2,3)}$ "simuliert". Bei einer Zuordnung der Aktionen $\alpha_{1,(2,2)}$ zu $\alpha_{1,2}$ und $\alpha_{2,(2,2)}$ zu $\alpha_{2,2}$ sowie der Abbildung von $\alpha_{1,(2,1)}$, $\alpha_{1,(2,3)}$, $\alpha_{2,(2,1)}$, $\alpha_{2,(2,3)}$ auf die leere Folge λ, sind in Q genau die gleichen Berechnungsfolgen ausführbar wie in P. Für jeden aktiven Ablauf α von P und den dazu bzgl. der getroffenen Aktionen-Zuordnung äquivalenten Ablauf β von Q gilt:

$$gv_2[val_P(\alpha)] = gv_2[val_Q(\beta)],$$

d.h. P und Q berechnen auch die selbe Funktion.

Systemname	Φ	Θ	Operations-bezeichnung
PV	$sv_j \geq 1$ true	$sv_j := sv_j - 1;$ $sv_j := sv_j + 1;$	$P(sv_j);$ $V(sv_j);$
PV two way bounded, $k>0$	$sv_j > 0$ $sv_j < k$	$sv_j := sv_j - 1;$ $sv_j := sv_j + 1;$	$P_k(sv_j);$ $V_k(sv_j);$
PV chunk, $t>0$	$sv_j \geq t$ true	$sv_j := sv_j - t;$ $sv_j := sv_j + t;$	$P(sv_j : t);$ $V(sv_j : t);$
PV multiple, $J = \{1,...,k\}$	$\forall j[j \in J \rightarrow sv_{i_j} \geq 1]$ true	$\forall j[j \in J \rightarrow sv_{i_j} = sv_{i_j} - 1];$ $\forall j[j \in J \rightarrow sv_{i_j} = sv_{i_j} + 1];$	$P(\{sv_{i_j} \mid j \in J\});$ $V(\{sv_{i_j} \mid j \in J\});$
PV general, $J = \{1,...,k\}$, $\forall j[j \in J \rightarrow t_j > 0]$	$\forall j[j \in J \rightarrow sv_{i_j} \geq t_j]$ true	$\forall j[j \in J \rightarrow sv_{i_j} = sv_{i_j} - t_j];$ $\forall j[j \in J \rightarrow sv_{i_j} = sv_{i_j} + t_j];$	$P(\{sv_{i_j} : t_j \mid j \in J\});$ $V(\{sv_{i_j} : t_j \mid j \in J\});$
up/down	$\sum_{r=1}^{k} sv_{i_r} \geq 0$ $\sum_{r=1}^{k} sv_{i_r} \geq 0$	$sv_j := sv_j + 1;$ $sv_j := sv_j - 1;$	$\{sv_{i_r} \mid 1 \leq r \leq k\}$: up($sv_j$); $\{sv_{i_r} \mid 1 \leq r \leq k\}$: down($sv_j$);

Tabelle 2.1.2.4/1: Für die Realisierung von Betriebssystemen wichtige Γ-Synchronisationssysteme, [Hof84]

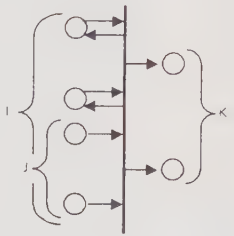
Systemname	Φ	Θ	Operations-bezeichnung
VRS (Vektorersetzung) $e_1, e_2 \in Z^m$, $sv:: = (sv_1,\ldots,sv_m)$;	$sv + e_1 \geq 0$	$sv: = sv + e_2$;	$<e_1 \mid e_2>$
VAS (Vektoraddition) $e \in Z^m$, $sv:: = (sv_1, \ldots, sv_m)$;	$sv + e \geq 0$	$sv: = sv + e$;	$<e>$
PN (Petrinetz) I, J, $K \subseteq \{1,\ldots,m\}$, $I \cap K = \emptyset$, $J \subseteq I$, $sv:: = (sv_1,\ldots,sv_m)$;	$\forall i[i \in I \rightarrow sv_i \geq 1]$	$\forall j \in J$: $svj: = svj-1$; $\forall k \in K$: $sv_k: = sv_k + 1$;	

Tabelle 2.1.2.4/2: Für theoretische Untersuchungen wichtige Γ-Synchronisationssysteme, [Hof84]

Anfangswerte: $sv_1 := 1;$ $sv_2 := 0;$ $gv_2 := 0;$ $gv_3 := 3;$
$bz_1 := 1;$ $lv_{1,1} := 0;$ $bz_2 := 1;$ $lv_{2,1} := 0;$

Prozeß Q_1:

$a_{1,1}$: *when* $(bz_1 = adr(a_{1,1}))$ *do*
$bz_1 := (if\ (lv_{1,1} \geq gv_3)\ then\ adr(a_{1,4})\ else\ adr(a_{1,2}));$
od;

$a_{1,(2,1)}$: $P(sv_1);$
$a_{1,(2,2)}$: *when* $(bz_1 = adr(a_{1,(2,2)}))$ *do*
$gv_2 := gv_2 + 1;$
$bz_1 := adr(a_{1,(2,3)});$
od;
$a_{1,(2,3)}$: $V(sv_2);$

$a_{1,3}$: *when* $(bz_1 = adr(a_{1,3}))$ *do*
$lv_{1,1} := lv_{1,1} + 1;$
$bz_1 := adr(a_{1,1});$
od;

$a_{1,4}$: *when* $(bz_1 = adr(a_{1,4}))$ *do*
exit$(P_1);$
od;

Prozeß Q_2:

$a_{2,1}$: *when* $(bz_2 = adr(a_{2,1}))$ *do*
$bz_2 := (if\ (lv_{2,1} \geq gv_3)\ then\ adr(a_{2,4})\ else\ adr(a_{2,2}));$
od;

$a_{2,(2,1)}$: $P(sv_2);$
$a_{2,(2,2)}$: *when* $(bz_2 = adr(a_{2,(2,2)}))$ *do*
$gv_2 := gv_2 * gv_2;$
$bz_2 := adr(a_{2,(2,3)});$
od;
$a_{2,(2,3)}$: $V(sv_1);$

$a_{2,3}$: *when* $(bz_2 = adr(a_{2,3}))$ *do*
$lv_{2,1} := lv_{2,1} + 1;$
$bz_2 := adr(a_{2,1});$
od;

$a_{2,4}$: *when* $(bz_2 = adr(a_{2,4}))$ *do*
exit$(P_2);$
od;

Bild 2.1.2.4/1: Beispiel-PV-Synchronisationssystem Q

2.1.3. Aktions- und Prozeßabhängigkeiten

Die vorangehend eingeführte Unterscheidung verschiedener Aktionentypen ist Voraussetzung für eine präzise Klassifizierung von Aktions- und Prozeßabhängigkeiten. Diese wiederum ist von entscheidender Bedeutung für die Beantwortung der Frage, ob eine im Fehlerfall erforderlich werdende Fehlerbehandlung auf den/die Fehlerverursacher beschränkt werden kann oder nicht. Auf der Grundlage einer elementaren *basiert auf Relation* zwischen lesenden und schreibenden Aktionen können sowohl *lokale und globale Abhängigkeiten*, als auch *kommunikationsbedingte* und *koordinierungsbedingte Abhängigkeiten* unterschieden werden. Lipton's Prozeßsystem-Modell bildet auch in diesem Fall eine gute Basis zur Formalisierung.

2.1.3.1. Basiert auf Relation

Unter Bezugnahme auf die im vorangehenden Abschnitt getroffenen Vereinbarungen lassen sich verschiedene Abhängigkeiten zwischen Aktionen einer aktiven Berechnungsfolge (und damit auch zwischen Prozessen eines Prozeßsystems) unterscheiden
Die Unterscheidung erfolgt auf der Grundlage eines elementaren Beziehungstyps zwischen lesenden und schreibenden Aktionen, der sich mittels der wie folgt definierten und jeder aktiven Berechnungsfolge α eines Prozeßsystems P zugeordneten *basiert auf Relation* B_α präzise beschreiben läßt.

Definition 2.1.3.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge von P und $\alpha[i],\alpha[j]\in W(\alpha)$ Dann *basiert* $\alpha[i]$ *in der Berechnungsfolge* α *auf* $\alpha[j]$ genau dann, wenn gilt:

$$i>j \quad \& \quad \exists v\in LV_{\alpha[i]}\cap SV_{\alpha[j]}\ \neg\exists\alpha[k]\in W(\alpha)\ [i>k>j\ \&\ v\in SV_{\alpha[k]}].$$

Kurzschreibweise: $\alpha[i]B_\alpha\alpha[j]$ ::<=> $\alpha[i]$ basiert (in α) auf $\alpha[j]$.

Anschaulich interpretiert basiert eine Aktion $\alpha[i]$ innerhalb einer Berechnungsfolge α auf einer Aktion $\alpha[j]$ immer genau dann, wenn $\alpha[i]$ ein Datum gelesen hat, das von $\alpha[j]$ stammt, d.h. das von $\alpha[j]$ zuletzt bearbeitet (verändert) wurde.

Beispiel 2.1.3.1/1: Die Berechnungsfolge $\alpha::=a_{1,1}a_{1,2}a_{1,3}a_{2,1}a_{2,2}a_{2,3}$ ist ein aktiver Ablauf innerhalb des Beispiel-Prozeßsystems P des Bildes 2.1.1.4/1. Zwischen den Aktionen $\alpha[i]\in W(\alpha)$, $i\in\{1,\ldots,6\}$, bestehen bzgl. der Relation B_α die folgenden Beziehungen:

$\alpha[2]=a_{1,2}B_\alpha a_{1,1}=\alpha[1]$ ist durch den Befehlszähler bz_1 bedingt,
$\alpha[3]=a_{1,3}B_\alpha a_{1,2}=\alpha[2]$ ist durch den Befehlszähler bz_1 bedingt,
$\alpha[5]=a_{2,2}B_\alpha a_{2,1}=\alpha[4]$ ist durch den Befehlszähler bz_2 bedingt,
$\alpha[6]=a_{2,3}B_\alpha a_{2,2}=\alpha[5]$ ist durch den Befehlszähler bz_2 bedingt,
$\alpha[5]=a_{2,2}B_\alpha a_{1,2}=\alpha[2]$ ist durch die globalen Variablen gv_1 und gv_2 bedingt.

Weitere Beziehungen zwischen Aktionen aus α bestehen nicht.

Das Bestehen der Relation B_α zwischen zwei Aktionen $\alpha[i]$ und $\alpha[j]$ deutet ein Abhängigkeitsverhältnis dieser beiden Aktionen an. Diese Abhängigkeitsbeziehung überträgt sich natürlich auch entsprechend auf die Prozesse $P_{bz(\alpha[i])}$ und $P_{bz(\alpha[j])}$, denen die Aktionen $\alpha[i]$ und $\alpha[j]$ angehören, und sie ist im Rahmen einer Fehlerbehandlung wie folgt zu berücksichtigen:

(1) Wird $\alpha[j]$ zurückgesetzt, so müssen im allgemeinen die von $\alpha[j]$ durchgeführten Änderungen an von $\alpha[i]$ gelesenen Variablen rückgängig gemacht werden, damit der nach Rücksetzung von $\alpha[j]$ erreichte Zustand in einem noch zu präzisierenden Sinne korrekt (konsistent) ist. D.h. aber, daß die von $\alpha[i]$ gelesene Information und damit auch die Ausführungsgrundlage für $\alpha[i]$ entzogen werden; eine Rücksetzung von $\alpha[i]$ ist daher in der Regel nicht zu vermeiden.

(2) Führt die Ausführung von $\alpha[j]$ zu einem fehlerhaften Ergebnis, so kann sich dieser Fehler auf $\alpha[i]$ fortpflanzen, d.h. im Falle eines Fehlers innerhalb von $\alpha[i]$ oder durch $\alpha[i]$ bedingt, muß $\alpha[j]$ als potentieller Fehlerverursacher in Betracht gezogen werden.

Anhand der Prozeßzugehörigkeit der Aktionen, die in Relation B_α zueinander stehen sowie des Typs der Variablen, über welche die Relation B_α zwischen zwei Aktionen $\alpha[i]$ und $\alpha[j]$ einer Berechnung α begründet wird, lassen sich Abhängigkeiten verschiedener Qualität unterscheiden. Wie noch zu zeigen sein wird, hängt es von der Art einer bestehenden Abhängigkeit ab, ob eine erforderlich werdende Fehlerbehandlung lokal auf einen Prozeß beschränkt bleiben kann oder nicht.

2.1.3.2. Lokale und globale Abhängigkeiten

Eine mögliche Klassifizierung von Abhängigkeiten stellt die Unterscheidung in *lokale* und *globale Abhängigkeiten* dar.

Definition 2.1.3.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge von P und $\alpha[i],\alpha[j]\in W(\alpha)$.

(1) $\alpha[i]$ ist von $\alpha[j]$ genau dann ***lokal abhängig***, wenn gilt:

$$\alpha[i]B_a\alpha[j] \;\&\; proc(\alpha[i],\alpha[j]).$$

Kurzschreibweise $l(\alpha[i]B_a\alpha[j])=\textbf{true}$ ::<=> $\alpha[i]$ ist lokal abhängig von $\alpha[j]$.

(2) $\alpha[i]$ ist von $\alpha[j]$ genau dann ***global abhängig***, wenn gilt:

$$\alpha[i]B_a\alpha[j] \;\&\; \neg proc(\alpha[i],\alpha[j]).$$

Kurzschreibweise: $g(\alpha[i]B_a\alpha[j])=\textbf{true}$ ::<=> $\alpha[i]$ ist global abhängig von $\alpha[j]$.

Beispiel 2.1.3.2/1: Zwischen den Aktionen der aktiven Berechnungsfolge α, $\alpha'':=a_{1,1}a_{1,2}a_{1,3}a_{2,1}a_{2,2}a_{2,3}$, des Beispiel-Prozeßsystems P (vgl. Bild 2.1.1.4/1) bestehen die folgenden lokalen Abhängigkeiten:

$$\alpha[2]B_a\alpha[1],\ \alpha[3]B_a\alpha[2],\ \alpha[5]B_a\alpha[4],\ \alpha[6]B_a\alpha[5].$$

Bei der basiert auf Relation zwischen $\alpha[5]$ und $\alpha[2]$ handelt es sich um eine globale Abhängigkeit: $\alpha[5]B_a\alpha[2]$.

Definition 2.1.3.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen, α sei eine aktive Berechnungsfolge von P. Durch die Berechnungsfolge α wird genau dann eine *Prozeßabhängigkeit* von Prozeß P_i zu Prozeß P_j, $i,j\in\{1,\ldots,n\}$, begründet, wenn gilt:

$$\exists\alpha[k],\alpha[l]\in W(\alpha):\ \alpha[k]B_a\alpha[l] \;\&\; i=bz(\alpha[k])\neq bz(\alpha[l])=j$$

oder äquivalent dazu:

$$\exists\alpha[k],\alpha[l]\in W(\alpha):\ g(\alpha[k]B_a\alpha[l])=\textbf{true}\ \&\ i=bz(\alpha[k])\ \&\ j=bz(\alpha[l]).$$

Beispiel 2.1.3.2/2: Durch die Berechnungsfolge $\alpha'':=a_{1,1}a_{1,2}a_{1,3}a_{2,1}a_{2,2}a_{2,3}$ des Beispiel-Prozeßsystems P des Bildes 2.1.1.4/1 wird eine Prozeßabhängigkeit von Prozeß P_2 zu Prozeß P_1 über die Beziehung $\alpha[5]=a_{2,2}B_aa_{1,2}=\alpha[2]$ begründet.

Das Bestehen von Abhängigkeiten zwischen Prozessen eines Prozeßsystems ist Voraussetzung für eine im Fehlerfall nicht ausschließlich auf einen Prozeß beschränkbare Fehlerbehandlung.
Wie noch zu zeigen sein wird, führt die Existenz von Prozeßabhängigkeiten jedoch nicht zwangsläufig zu einem in Fehlersituationen nicht exklusiv auf einen Prozeß begrenzbaren Recovery-Aufwand. Beispielsweise kann ausreichendes a priori Wissen

über den Sinn und Zweck einer Prozeß-Prozeß Beziehung eine Reduzierung des im Fehlerfall erforderlichen Fehlerbehandlungsaufwandes gestatten.

2.1.3.3. Kommunikationsbedingte Abhängigkeiten

Eine Art von Prozeß-Prozeß Beziehungen, die zu Prozeßsystem-Zuständen führen können, in welchen trotz des Vorhandenseins globaler Abhängigkeiten eine ausschließlich auf die als fehlerhaft erkannten Prozesse einschränkbare Fehlerbehandlung möglich ist, sind *kommunikationsbedingte Abhängigkeiten*. Als Abhängigkeiten dieses Typs werden Beziehungen zwischen nachrichtenübertragenden Aktionen (siehe Definition 2.1.2.3/1-(2)) bezeichnet.

Definition 2.1.3.3/1: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge von P und $\alpha[i],\alpha[j] \in W(\alpha)$. Dann ist $\alpha[i]$ *kommunikationsbedingt abhängig von* $\alpha[j]$ genau dann, wenn gilt:

$$\alpha[i] B_\alpha \alpha[j] \;\&\; n(\alpha[i]) \;\&\; n(\alpha[j]) \;\&\; \exists np \in V_{\alpha[i]} \cap V_{\alpha[j]}.$$

Kurzschreibweise: $n(\alpha[i] B_\alpha \alpha[j])$=**true** ::<=> $\alpha[i]$ ist kommunikationsbedingt abhängig von $\alpha[j]$.

Beispiel 2.1.3.3/1: Die Folge $\alpha ::= a_{1,1} a_{1,(2,1)} a_{1,(2,2)} a_{1,(2,3)} a_{1,3} a_{2,1} a_{2,(2,1)} a_{2,(2,2)} a_{2,(2,3)} a_{2,3}$ stellt einen aktiven Ablauf innerhalb des Beispiel-FIFO-Nachrichtensystems Q (vgl. Bild 2.1.2.3/1) dar. Kommunikationsbedingt abhängig sind die Aktionen $\alpha[9]=a_{2,(2,3)}$ (von $\alpha[2]=a_{1,(2,1)}$) und $\alpha[7]=a_{2,(2,1)}$ (von $\alpha[4]=a_{1,(2,3)}$).
Nicht kommunikationsbedingt abhängig von Aktion $\alpha[2]=a_{1,(2,1)}$ ist jedoch etwa die Aktion $\alpha[4]=a_{1,(2,3)}$ oder die Aktion $\alpha[7]=a_{2,(2,1)}$, denn die Aktionen $\alpha[2]$ und $\alpha[4]$ bzw. $\alpha[2]$ und $\alpha[7]$ beziehen sich auf unterschiedliche Nachrichtenpuffer, d.h.:

$$\neg \exists np \in V_{\alpha[2]} \cap V_{\alpha[4]} \text{ und } \neg \exists np \in V_{\alpha[2]} \cap V_{\alpha[7]}.$$

Bemerkung: Aufgrund der Struktur einer nachrichtenübertragenden Aktion ergibt es sich, daß ein kommunikationsbedingtes Abhängigkeitsverhältnis über die Variablen b_i und f_i, die den Füllungsgrad und die freie Kapazität eines Nachrichtenpuffers anzeigen, auch zwischen zwei *send*- oder zwei *receive*-Aktionen entstehen kann.

2.1.3.4. Koordinierungsbedingte Abhängigkeiten

Eine rein prozeßlokale Fehlerbeseitigung ohne Berücksichtigung globaler Abhängigkeiten ist häufig auch im Falle von *koordinierungsbedingten Abhängigkeiten*,

wie sie auf der Basis von Beziehungen zwischen synchronisierenden Aktionen (vgl. Definition 2.1.2.4/1-(2)) zwischen Prozessen eines Prozeßsystems entstehen können, möglich.

Definition 2.1.3.4/1: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge von P und $\alpha[i],\alpha[j]\in W(\alpha)$. Dann ist $\alpha[i]$ *koordinierungsbedingt abhängig von* $\alpha[j]$, genau dann, wenn gilt:

$$\alpha[i]B_a\alpha[j] \;\&\; s(\alpha[i]) \;\&\; s(\alpha[j]).$$

Kurzschreibweise: $s(\alpha[i]B_a\alpha[j])$=**true** ::<=> $\alpha[i]$ ist koordinierungsbedingt abhängig von $\alpha[j]$.

Beispiel 2.1.3.4/1: Zwischen den Aktionen der wie folgt definierten aktiven Berechnungsfolge α,

$$\alpha ::= \alpha_{1,1}\alpha_{1,(2,1)}\alpha_{1,(2,2)}\alpha_{1,(2,3)}\alpha_{1,3}\alpha_{2,1}\alpha_{2,(2,1)}\alpha_{2,(2,2)}\alpha_{2,(2,3)}\alpha_{2,3},$$

des Beispiel-PV-Synchronisationssystems Q (siehe Bild 2.1.2.4/1) bestehen die folgenden koordinierungsbedingten Abhängigkeiten:

$$\alpha[9]=\alpha_{2,(2,3)}B_a\alpha_{1,(2,1)}=\alpha[2] \text{ und } \alpha[7]=\alpha_{2,(2,1)}B_a\alpha_{1,(2,3)}=\alpha[4].$$

2.1.4. Prozeßsystemklassen

Gemäß dem Typ der unterschiedenen Abhängigkeiten, die zwischen Aktionen aktiver Berechnungsabläufe eines Prozeßsystems entstehen können, kann die Menge aller Prozeßsysteme in die vier folgenden, sich teilweise überlappenden Prozeßsystemklassen unterteilt werden:

(1) in *Systeme unabhängiger Prozesse*,
(2) in *Systeme kooperierender Prozesse*,
(3) in *Systeme kommunizierender Prozesse* und
(4) in *Systeme koordinierender Prozesse*.

Ähnliche Klassifizierungen findet man auch in der Literatur, so etwa in [ALB1], wieder.

2.1.4.1. Systeme unabhängiger Prozesse

Systeme (voneinander) unabhängiger Prozesse zeichnen sich dadurch aus, daß sich in ihnen nur lokale Abhängigkeitsbeziehungen zwischen Aktionen bilden können, d.h. zwischen zwei Prozessen P_i und P_j, $i \neq j$, eines Systems unabhängiger Prozesse kann kein Informationsfluß entstehen, was insbesondere auch bedeutet, daß ein in P_i (bzw. P_j) aufgetretener Fehler sich nicht mittels eines Berechnungsablaufs α von P_i (bzw. P_j) nach P_j (P_i) fortpflanzen kann. Wie noch zu zeigen sein wird, läßt sich bei Systemen dieses Typs deshalb eine erforderlich werdende Fehlerbehandlung prinzipiell immer auf diejenigen Prozesse beschränken, die als fehlerverursachend identifiziert werden. *Prinzipiell* heißt: Bei Zugrundelegung eines entsprechenden Fehlerbehandlungs-Modells (siehe Abschnitt 2.2).

Definition 2.1.4.1/1: Ein Prozeßsystem $P=(A,D,w)$ wird als ein *System (voneinander) unabhängiger Prozesse* bezeichnet genau dann, wenn gilt:

$$\forall \alpha \forall \alpha[i],\alpha[j] \in W(\alpha)\ [(\alpha \text{ aktiv in } P\ \&\ \alpha[i] B_\alpha \alpha[j])\ \Rightarrow\ l(\alpha[i] B_\alpha \alpha[j])].$$

Kurzschreibweise: $l(P)$=**true** ::<=> P ist ein System voneinander unabhängiger Prozesse.

Beispiel 2.1.4.1/1: Das in Bild 2.1.1.4/1 dargestellte Beispiel-Prozeßsystem $P=(A,D,w)$ ist kein System unabhängiger Prozesse, im Gegensatz zu dem aus P durch Löschung einer der Aktionen $\alpha_{1,2}$ oder $\alpha_{2,2}$ ableitbaren Prozeßsystem $P'=(A',D,w)$.

Definition 2.1.4.1/2: Die Klasse der Systeme unabhängiger Prozesse $CL_4(P)$ ist wie folgt definiert:

$$CL_4(P)::=\{\ P \mid P=(A,D,w) \text{ ist Prozeßsystem } \&\ l(P)=\textbf{true}\}.$$

2.1.4.2. Systeme kooperierender Prozesse

Systemen (miteinander) kooperierender Prozesse ist eigen, daß in ihnen Prozeß-Prozeß Beziehungen vielfältigster Art bestehen können. Dies macht im Fehlerfall eine im allgemeinen nicht nur auf die fehlerverursachenden Prozesse beschränkbare Fehlerbehandlung notwendig.

Definition 2.1.4.2/1: Ein Prozeßsystem $P=(A,D,w)$ wird als ein ***System (miteinander) kooperierender Prozesse*** bezeichnet genau dann, wenn gilt:

$$\dot{\exists}\alpha\dot{\exists}\alpha[i],\alpha[j]\in W(\alpha)\ [\alpha \text{ aktiv in } P\ \&\ g(\alpha[i]B_a\alpha[j])].$$

Kurzschreibweise $g(P)=\textbf{true}$ ::<=> P ist ein System miteinander kooperierender Prozesse.

Beispiel 2.1.4.2/1: Die Prozeßsysteme der Bilder 2.1.1.4/1, 2.1.2.3/1 und 2.1.2.4/1 gehören allesamt zur Klasse der Systeme kooperierender Prozesse.

Definition 2.1.4.2/2: Die Klasse der Systeme kooperierender Prozesse $CL_g(P)$ kann wie folgt charakterisiert werden:

$$CL_g(P)::=\{\ P \mid P=(A,D,w) \text{ ist Prozeßsystem } \&\ g(P)=\textbf{true}\}.$$

Satz 2.1.4.2/1: Die Prozeßsystemklassen $CL_4(P)$ und $CL_g(P)$ stehen wie folgt in Beziehung:

(1) $CL_4(P)\cap CL_g(P)=\phi$ und
(2) $CL_4(P)\cup CL_g(P)=CL(P)$,

wobei ϕ die leere Menge bezeichnet und $CL(P)$. $=\{\ P \mid P=(A,D,w)$ ist Prozeßsystem$\}$ die Menge aller Prozeßsysteme repräsentiert.

Beweis: Folgt unmittelbar mit den Definitionen 2.1.4.1/1,2 und 2.1.4.2/1,2. ■

2.1.4.3. Systeme kommunizierender Prozesse

Systeme (miteinander) kommunizierender Prozesse sind dadurch charakterisiert, daß in ihnen Abhängigkeitsbeziehungen zwischen Prozessen nur über einen bestimmten Typ von Aktionen entstehen können.
Systeme kommunizierender Prozesse gehören der Klasse der kooperierenden Prozesse an. Aufgrund der weitgehend bekannten Semantik nachrichtenübertragender Aktionen lassen derartige Systeme, wie noch zu zeigen sein wird, in vielen Fällen eine weniger aufwendige Fehlerbehandlung zu

Definition 2.1.4.3/1: Ein Prozeßsystem $P=(A,D,w)$ wird als ein *System (miteinander) kommunizierender Prozesse* bezeichnet genau dann, wenn:

$$\forall\alpha\forall\alpha[i],\alpha[j]\in W(\alpha)\,[(\alpha \text{ aktiv in } P \;\&\; g(\alpha[i]B_a\alpha[j])) \Rightarrow n(\alpha[i]B_a\alpha[j])]$$
$$\&\;\exists\alpha\exists\alpha[i],\alpha[j]\in W(\alpha)\,[\alpha \text{ aktiv in } P \;\&\; g(\alpha[i]B_a\alpha[j])].$$

Kurzschreibweise $n(P)$=**true** $\Leftrightarrow$ P ist ein System miteinander kommunizierender Prozesse.

Definition 2.1.4.3/2: Die Klasse der Systeme kommunizierender Prozesse $CL_n(P)$ ist wie folgt definiert:

$$CL_n(P) ::= \{\, P \mid P=(A,D,w) \text{ ist Prozeßsystem } \&\; n(P)=\textbf{true}\}.$$

Beispiel 2.1.4.3/1: Das in Bild 2.1.2.3/1 dargestellte Beispiel-FIFO-Nachrichtensystem gehört nicht der Klasse kommunizierender Prozesse an, da in der Berechnugsfolge α,

$$\alpha_{..}=\alpha_{1,1}\alpha_{1,(2,1)}\alpha_{1,(2,2)}\alpha_{1,(2,3)}\alpha_{1,3}\alpha_{2,1}\alpha_{2,(2,1)}\alpha_{2,(2,2)}\alpha_{2,(2,3)}\alpha_{2,3},$$

zwischen den Aktionen $\alpha[3]$ und $\alpha[8]$ eine globale Beziehung besteht, für die $n(\alpha[8]B_a\alpha[3])$=**false** gilt.
Bei Entfernung einer der beiden Aktionen $\alpha_{1,(2,2)}$ oder $\alpha_{2,(2,2)}$ aus der Aktionenmenge A entsteht ein Prozeßsystem Q', für das $Q'\in CL_n(P)$ gilt.

Satz 2.1.4.3/1: Die Prozeßsystemklassen $CL_g(P)$ und $CL_n(P)$ stehen in folgender Beziehung:

$$CL_n(P) \subset CL_g(P),$$

d.h. jede Aussage über Prozeßsysteme aus $CL_g(P)$ gilt auch für Prozeßsysteme aus $CL_n(P)$.

Beweis: $CL_n(P)\subset CL_g(P)$ folgt mit den Definitionen 2.1.4.2/1,2 und 2.1.4.3/1,2. Die

Echtheit der Inklusion zeigt Beispiel 2.1.4.3/1, denn, wie erläutert, ist das FIFO-Nachrichtensystem Q, auf das in diesem Beispiel bezug genommen wird, kein System (ausschließlich) kommunizierender Prozesse, gemäß Definition 2.1.4.2/1 jedoch ein System kooperierender Prozesse. ■

Satz 2.1.4.3/2: Nicht jedes System $P=(A,D,w)$ kommunizierender Prozesse ist ein Γ-Nachrichtensystem und umgekehrt

Beweis: Das Γ-Nachrichtensystem Q des Bildes 2.1.2.3/1 ist gemäß den Erläuterungen zu Beispiel 2.1.4.3/1 kein System (ausschließlich) kommunizierender Prozesse.
Auf der anderen Seite lassen sich auf einfache Weise Systeme kommunizierender Prozesse konstruieren, die nicht nur aus Aktionen des Typs bestehen wie nach Definition 2.1.2.3/1-(2) für Γ-Nachrichtensysteme zulässig. ■

Bemerkung: Trotz Satz 2.1.4.3/2 läßt sich feststellen: Systeme kommunizierender Prozesse entsprechen Γ-Nachrichtensystemen, die bzgl. ihres dynamischen Verhaltens auf nachrichtenaustauschende Beziehungen zwischen Prozessen eingeschränkt sind

2.1.4.4. Systeme koordinierender Prozesse

Systeme (sich gegenseitig) koordinierender Prozesse besitzen die Eigenschaft, daß sich in ihnen Prozeßbeziehungen nur mittels synchronisierender Aktionen realisieren lassen. Auch sie gehören der Klasse kooperierender Prozesse an. Im Gegensatz zum allgemeinen Fall von Systemen kooperierender Prozesse reicht das bzgl. der Bedeutung synchronisierender Aktionen vorhandene a priori Wissen gelegentlich aber aus, um in Systemen koordinierender Prozesse aufgetretene Fehler auf, wie noch zu zeigen sein wird, einfache Art zu beseitigen.

Definition 2.1.4.4/1: Ein Prozeßsystem $P=(A,D,w)$ wird als ein *System (sich gegenseitig) koordinierender Prozesse* bezeichnet genau dann, wenn:

$$\forall\alpha\forall\alpha[i],\alpha[j]\in W(\alpha)\,[(\alpha \text{ aktiv in } P \;\&\; g(\alpha[i]B_a\alpha[j])) \Rightarrow s(\alpha[i]B_a\alpha[j])]$$
$$\&\;\; \dot{\exists}\alpha\dot{\exists}\alpha[i],\alpha[j]\in W(\alpha)\,[\alpha \text{ aktiv in } P \;\&\; g(\alpha[i]B_a\alpha[j])].$$

Kurzschreibweise: $s(P)=\mathbf{true}$.:<=> P ist ein System koordinierender Prozesse.

Beispiel 2.1.4.4/1: Das in Bild 2.1.2.4/1 skizzierte PV-Synchronisationssystem Q ist ein System kooperierender Prozesse (vgl. Definition 2.1.4.2/1), aber kein System (ausschließlich) koordinierender Prozesse, denn zwischen den Aktionen $\alpha[3]$ und $\alpha[8]$ der aktiven Berechnungsfolge α,

$$\alpha ::= a_{1,1}a_{1,(2,1)}a_{1,(2,2)}a_{1,(2,3)}a_{1,3}a_{2,1}a_{2,(2,1)}a_{2,(2,2)}a_{2,(2,3)}a_{2,3},$$

besteht die Beziehung $\alpha[8]=a_{2,(2,2)}B_a a_{1,(2,2)}=\alpha[3]$, für welche $g(\alpha[8]B_a\alpha[3])$=**true**, jedoch $s(\alpha[8]B_a\alpha[3])$=**false** gilt.
Durch Abzug einer der beiden Aktionen $a_{1,(2,2)}$ oder $a_{2,(2,2)}$ von der Aktionenmenge A von Q kann ein System Q' koordinierender Prozesse jedoch leicht konstruiert werden.

Definition 2.1.4.4/2: Die Klasse der Systeme koordinierender Prozesse $CL_s(P)$ ist wie folgt charakterisiert:

$$CL_s(P) ::= \{\ P \mid P=(A,D,w) \text{ ist Prozeßsystem } \&\ s(P)=\textbf{true}\},$$

Satz 2.1.4.4/1: Die Prozeßsystemklassen $CL_g(P)$ und $CL_s(P)$ stehen in der folgenden Beziehung zueinander:

$$CL_s(P) \subset CL_g(P),$$

d.h. jede Aussage über Systeme kooperierender Prozesse ist auf Systeme koordinierender Prozesse anwendbar.

Beweis: $CL_s(P) \subseteq CL_g(P)$ folgt mit den Definitionen 2.1.4.2/1,2 und 2.1.4.4/1,2. Die Echtheit der Inklusion zeigt Beispiel 2.1.4.4/1. ■

Satz 2.1.4.4/2: Nicht jedes System $P=(A,D,w)$ koordinierender Prozesse ist ein Γ-Synchronisationssystem und umgekehrt.

Beweis: Das in Bild 2.1.2.4/1 skizzierte PV-Synchronisationssystem ist, wie in Beispiel 2.1.4.4/1 erläutert, kein System (ausschließlich) koordinierender Prozesse. Andererseits können in Systemen koordinierender Prozesse auch Aktionen enthalten sein, die nicht Definition 2.1.2.4/1 für Γ-Synchronisationssysteme genügen. ■

Bemerkung: Unabhängig von der Aussage des Satzes 2.1.4.4/2 ist folgende Vorstellung zutreffend: Systeme koordinierender Prozesse können als Γ-Synchronisationssysteme betrachtet werden, denn ihr dynamisches Verhalten ist ausschließlich auf synchronisierende Prozeßbeziehungen beschränkt.

2.2. Fehlerbehandlungs-Modell

Das für die zu untersuchenden Fragestellungen wichtige Fehlerbehandlungs-Modell orientiert sich bzgl. der Modellierung des unterlagerten Rechensystems an Ergebnissen der Arbeiten von J.E.B. Moss, [Mos81], sowie B. Lampson und H. Sturgis, [LS81]. Hinsichtlich der verwendeten Bezeichnungsweisen und skizzierten Fehlerbehandlungs-Techniken wird auf Arbeiten von T. Anderson und P.A. Lee, [AL81], [AL82], J.C. Laprie, [Lap85], und A. Reuter, [Reu81], zurückgegriffen.

Nach der Vorstellung des dem Fehlerbehandlungs-Modell unterlagerten Rechensystems, einschließlich seines Verhaltens im Fehlerfall, wird die prinzipielle Vorgehensweise bei der Fehlerbehandlung in Prozeßsystemen erläutert, und es werden die zu ihrer Realisierung benötigten elementaren Mechanismen beschrieben. Der Ablauf einer Fehlerbehandlung wird schließlich anhand eines ausführlichen Beispiels skizziert, wobei in diesem Zusammenhang auch auf die beiden Phänomene der Rücksetz- und Fehlerausbreitung näher eingegangen wird.

2.2.1. Unterlagertes Rechensystem

Da in dieser Arbeit von der Fehlerbehandlung in Prozeßsystemen ein wirklichkeitsnahes Bild gezeichnet werden soll, das eine Umsetzung abgeleiteter theoretischer Ergebnisse in die Praxis zuläßt, genügt es nicht, die Betrachtung ausschließlich auf die Prozeßsystemebene zu beschränken. Vielmehr ist eine Modellierung des einem Prozeßsystem in der Realität unterlagerten Rechensystems mit seinen Systemkomponenten, der zugehörigen Prozeßsystem-Einbettung, seiner Funktionsweise und seines Verhaltens im Fehlerfall erforderlich. Die entsprechenden bezüglich eines unterlagerten Rechensystems zu fordernden Charakteristiken und zu treffenden Annahmen werden in den folgenden Abschnitten skizziert.

2.2.1.1. Systemkomponenten

Die zu unterscheidenden Systemkomponenten des in die Untersuchungen einbezogenen Rechensystem-Modells zeigt Bild 2.2.1.1/1 anhand der schematischen Darstellung eines *Monoprozessorsystems*.

Im einzelnen setzt sich das Rechensystem-Modell aus *fünf Systemkomponenten*, *vier Hardware-Bestandteilen* und der *Basissoftware* zusammen. Bei der Hardware handelt es sich im einzelnen um:

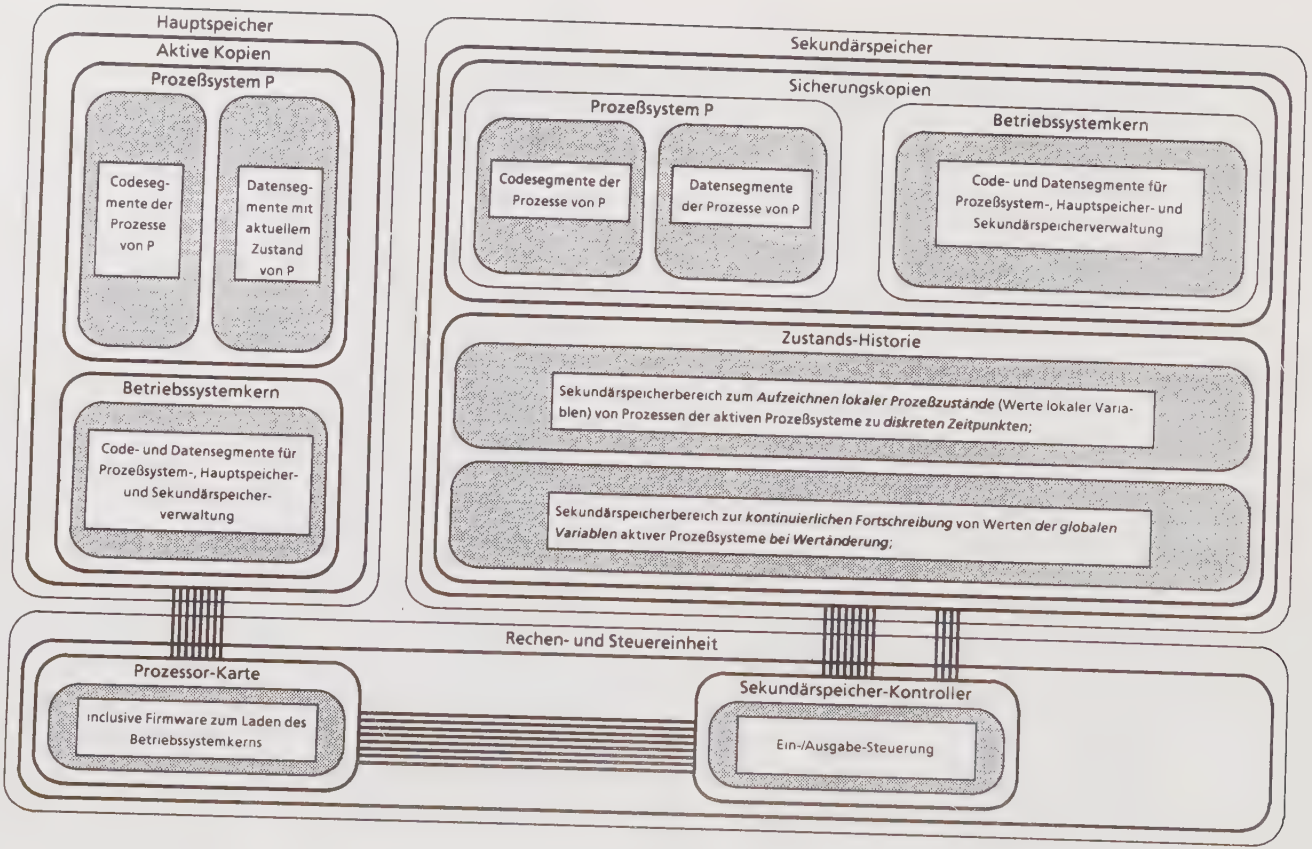


Bild 2.2.1.1/1: Monoprozessorsystem-Modell

(1) einen *Hauptspeichermodul*; er dient als Arbeitsspeicher des Rechensystems; in ihm werden Prozeßsysteme zur Ausführung gebracht;

(2) einem *Sekundärspeicher*; er übernimmt die Aufgabe der langfristigen Speicherung und Sicherung von Daten inclusive der Prozeßsysteme;

(3) einer *Rechen- und Steuereinheit*; diese ermöglicht die Bearbeitung der im System gespeicherten Daten, in dem sie die Ausführung der in den Arbeitsspeicher geladenen Prozeßsysteme und den Transfer von Daten zwischen Arbeits- und Sekundärspeicher steuert;

(4) *Kanal- und Busverbindungen*; über sie ist die für den Datenaustausch zwischen den Speichern erforderliche Kopplung von Hauptspeicher, Rechen-/Steuereinheit und Sekundärspeicher realisiert.

Die *Basissoftware*, die als Betriebssystemkern bezeichnet wird, hat zum einen die Aufgabe den Haupt- und Sekundärspeicher zu verwalten, zum anderen muß sie verschiedene Basismechanismen/Funktionen bereitstellen, die einen ordnungsgemäßen Ablauf von Applikationen (im vorliegenden Fall sind dies Prozeßsysteme) ermöglichen. Die vorausgesetzten Basismechanismen/Funktionen werden im Unterabschnitt 2.2.1.3, in dem die Funktionsweise des unterlagerten Rechensystems beschrieben wird, noch näher erläutert.

Das in Bild 2.2.1.1/1 skizzierte Rechensystem-Modell stellt ein zentral organisiertes System dar. Ausgehend von dieser Organisationsform kann ein verteiltes System (*Mehrrechnersystem*) durch Zusammenschluß mehrerer zentraler Systeme über ein Netzwerk, dessen Topologie für das folgende belanglos ist, und die Bereitstellung entsprechender Kommunikationsdienste im Betriebssystemkern konstruiert werden (Bild 2.2.1.1/2).
Schließlich kann, durch Anpassung der Basissoftware an die verteilte Systemumgebung, auf der Grundlage eines verteileten Rechensystems ein ***physisch zwar verteiltes, aber logisch zentralisiertes System*** konzipiert werden (Bild 2.2.1.1/3).
Ein Rechensystem ist physisch verteilt, jedoch logisch zentralisiert heißt: Aus der Sicht der auf ihm ablaufenden Anwendungen (hier: Prozeßsysteme) besteht kein Unterschied zu einem physisch und logisch zentral organisierten System; den einzelnen Betriebssystemkernen, nicht dem Systemimplementierer bleibt es in diesem Fall überlassen Code und Daten von Prozeßsystemen entsprechend der vorgegebenen Ziele (z.B. Maximierung des Systemdurchsatzes) im Netz zu verteilen.

Die bei der "logischen Zentralisierung" eines physisch verteilten Systems zu lösende Aufgabe besteht im wesentlichen darin, durch Einführung entsprechender Protokolle die parallelen Aktivitäten der einzelnen Betriebssystemkerne in der Weise aufeinander abzustimmen, daß an der Schnittstelle zum Anwender das Betriebssystemverhalten einem zentralen System gleicht. Die Entwicklung eines logisch zentralisiert

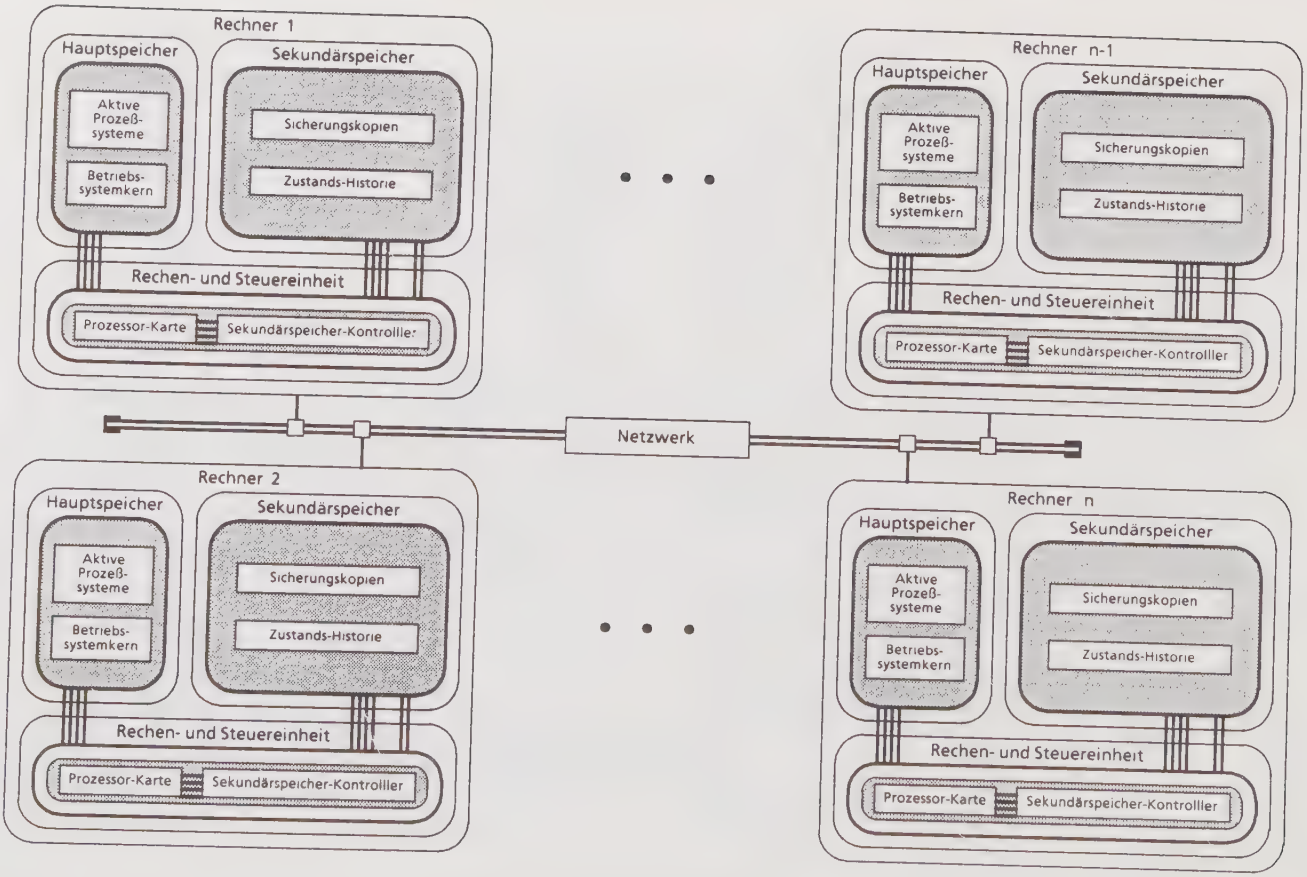


Bild 2.2.1.1/2: Mehrrechnersystem-Modell

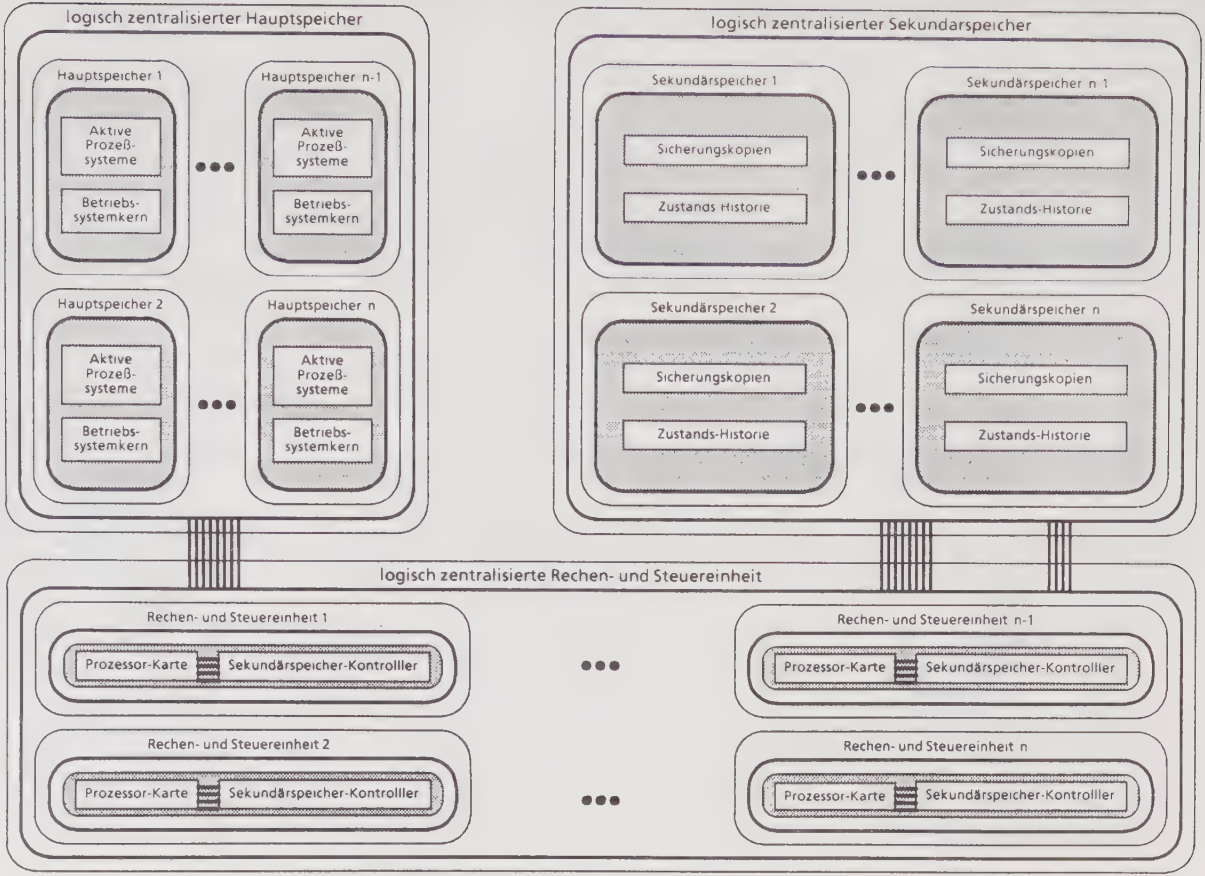


Bild 2.2.1.1/3: Logisch zentralisiertes Mehrrechnersystem

arbeitenden Betriebssystemkerns für ein *Mehrprozessorsystem* (Bild 2.2.1.1/4) erfordert ähnliche Aktivitäten.

Annahme 2.2.1.1/1: Im folgenden wird davon ausgegangen, daß alle Prozeßsystemen unterlagerten *Rechensysteme logisch zentralisiert* sind, d.h. gleichgültig, ob es sich physisch um Monoprozessorsysteme, Mehrrechnersysteme oder Mehrprozessorsysteme handelt, verhalten sich Prozeßsystemen unterlagerte Rechensysteme wie zentral organisierte Systeme.

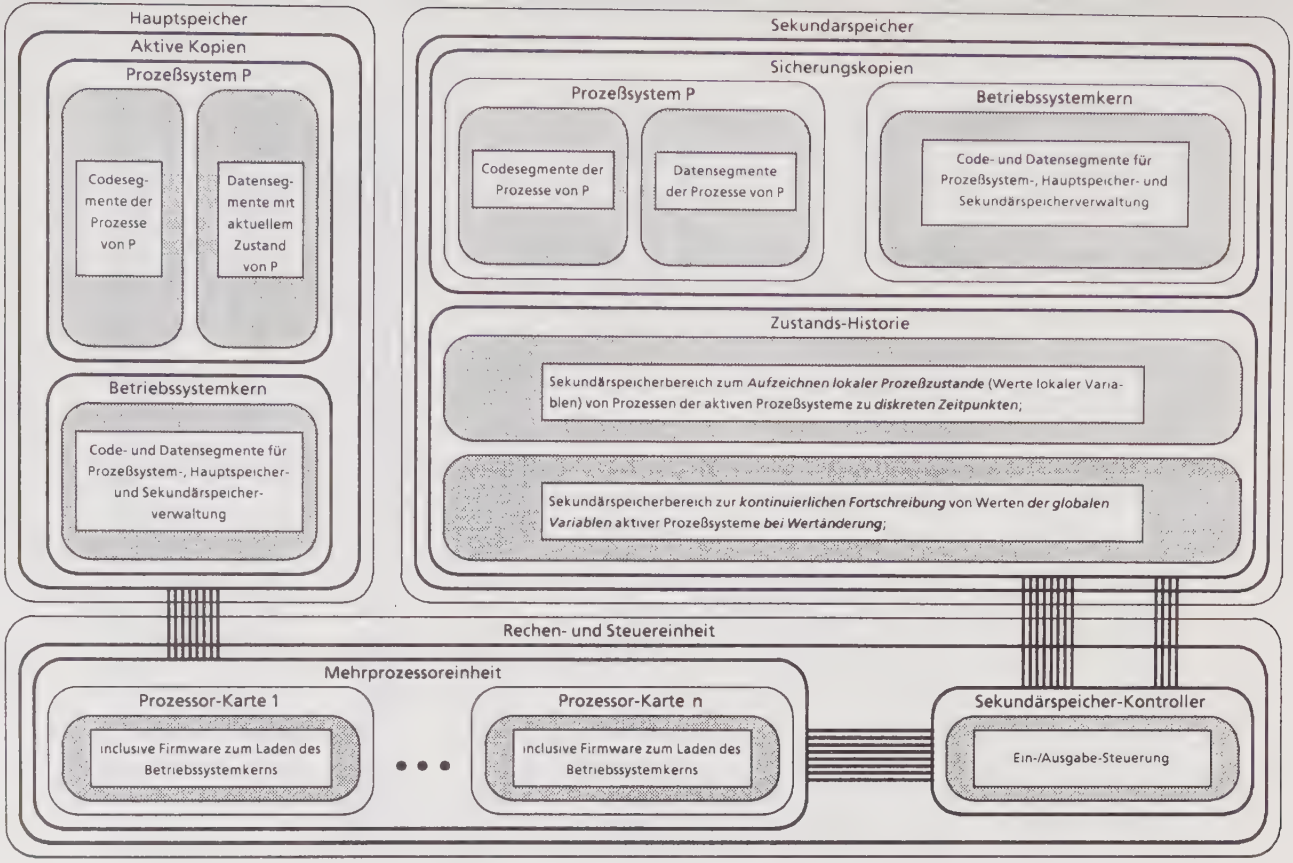


Bild 2.2.1.1/4: Mehrprozessorsystem-Modell

2.2.1.2. Prozeßsystemeinbettung

Prozeßsysteme werden in dem vorliegenden Rechensystem-Modell auf die folgende Weise verwaltet: Im passiven Zustand wird das Prozeßsystem auf dem Sekundärspeicher gehalten, und bei Aktivierung wird eine Kopie in den Hauptspeicher geladen und zur Ausführung gebracht.

Um die Prozeßsystemeinbettung einfach halten, anschaulich darstellen, bezüglich einer möglichen Realisierung aber trotzdem ausreichend konkretisieren zu können, werden folgende Annahmen gemacht:

Annahme 2.2.1.2/1: Jedem Prozeßsystem wird bei seiner Aktivierung ein eigener Adreßbereich im Hauptspeicher zugewiesen, d.h. die Adreßräume von Aktivierungen zweier beliebiger Prozeßsysteme P und Q sind disjunkt. Ein Fehler kann sich nicht von einer Prozeßsystem-Aktivierung auf eine andere fortpflanzen.

Kann Annahme 2.2.1.2/1 nicht eingehalten werden, da sich die Aktivierungen zweier Prozeßsysteme P und Q gegenseitig beeinflussen, so besteht eine Möglichkeit darin, die Annahme 2.2.1.2/1 durch Definition eines neuen Prozeßsystems P' als der Vereinigung von P und Q zu erfüllen, wobei dann in diesem Fall P' die Annahme 2.2.1.2/1 erfüllt. Vor einer Vereinigung von P und Q zu P' müssen die Aktionen und entsprechend auch die Variablen von P und Q gegebenenfalls in zwei disjunkte Mengen von Aktionen-/Variablennamen umbenannt werden; der Anfangszustand w_P von P' kann definiert werden als $w_P := (w_P, w_Q)$. Im Grenzfall führt auf diese Weise die Vereinigung aller Prozeßsysteme P eines Rechensystems zu einem einzigen Prozeßsystem P'. Dies rechtfertigt eine weitere Vereinfachung:

Annahme 2.2.1.2/2: Im folgenden werden ausschließlich Rechensysteme mit einem einzigen Prozeßsystem P betrachtet, das bei seiner Aktivierung vom Betriebssystemkern in den Hauptspeicher des Rechensystems geladen und dort zur Ausführung gebracht wird.

Eine analoge Überlegung (mehrere aktivierte Kopien eines Prozeßsystems P können zu einem Prozeßsystem P' zusammengefaßt werden) führt zu folgender Vereinfachung:

Annahme 2.2.1.2/3: Auf die Modellierung mehrfacher parallel ablaufender Aktivierungen eines Prozeßsystems kann im folgenden verzichtet werden, da unter der

Einschränkung einer einfachen Aktivierung eines Prozeßsystems abgeleitete Ergebnisse (wie angedeutet) problemlos verallgemeinert werden können.

2.2.1.3. Funktionsweise des Rechensystems

Mit dem Rechensystem-Modell wird folgende Funktionsweise implizit verbunden:

(1) Beim Systemstart, d.h. dem Einschalten des Rechensystems, wird der Betriebssystemkern mittels entsprechender Firmware in den Hauptspeicher geladen.

(2) Nach erfolgreichem Abschluß des Ladevorganges geht die Kontrolle auf den Betriebssystemkern über, der das (nach den Annahmen des vorangehenden Abschnitts einzige) Prozeßsystem P aktiviert und zum Ablauf bringt.

(3) Neben Mechanismen, welche die Atomarität von aktivierten Aktionen von P gewährleisten, stellt der Betriebssystemkern dem Prozeßsystem P synchronisierende Operationen zur Koordination von Aktionen parallel in Ausführung befindlicher Prozesse, nachrichtenübertragende Operationen zur Ermöglichung der Kommunikation zwischen Prozessen und Operationen zur Fortschreibung der mit einem Ablauf verbundenen Historie von Prozeßsystem-Zuständen zur Verfügung.

(4) Die Aufzeichnung der Zustands-Historie zu aktiven Berechnungen von P ermöglicht die Realisierung von im Betriebssystemkern verankerten Fehlerbehandlungs-Strategien. Diese sorgen im Fehlerfall für die Wiederherstellung eines konsistenten Prozeßsystem-Zustandes. Einzelheiten zur Fehlerbehandlung werden im Abschnitt 2.2.2 skizziert.

(5) Während seiner Aktivierung sind sämtliche vom Prozeßsystem P bearbeiteten Daten hauptspeicherresident, d.h. P bildet ein für sich abgeschlossenes System, in dem die Reversibilität von Änderungen aktiver Aktionenfolgen mittels der am Sekundärspeicher aufgezeichneten Zustands-Historie von der Konzeption her gewährleistet ist.

(6) Eine aktive Berechnung α des Prozeßsystems P führt zu einem als ordnungsgemäß zu bezeichnenden Rechensystem-Stop, wenn im Zustand $val_P(\alpha)$ die Menge der lauffähigen Aktionen $lf_P(\alpha)$ leer ist. Im Falle eines ordnungsgemäßen Rechensystem-Stops kann die am Sekundärspeicher gehaltene Zustandssicherungs-Information freigegeben werden.

(7) Auf das Verhalten im Fehlerfall, das zu nicht ordnungsgemäßen Rechensystem-Stops führen kann, geht der Unterabschnitt 2.2.1.4 näher ein.

Zur Erläuterung bleibt nachzutragen, daß im vorliegenden Modell (Benutzer-)Interaktionen mit einem Prozeßsystem nicht erfaßt werden, um die Abgeschlossenheit des Gesamtsystems und damit die prinzipielle Reversibilität von durch Aktionen bewirkten Zustandsänderungen garantieren zu können. Dies bedeutet jedoch keine

Einschränkung, da Interaktionen logisch ohne weiteres durch einen (mehrere) eigenständigen (eigenständige) zu einem Prozeßsystem hinzuzufügenden (hinzuzufügende) Prozeß (Prozesse) simuliert werden können.
Des weiteren ist zu beachten, daß die in der Beschreibung der Funktionsweise eines Rechensystems skizzierten Punkte (4) und (5) die Erfüllung der folgenden Annahme 2.2.1.3/1 voraussetzen und die Anforderungen an die im Betriebssystemkern verankerten Mechanismen zur Sicherung aktiver Prozeßsystemabläufe in Annahme 2.2.1.3/2 näher präzisiert werden:

Annahme 2.2.1.3/1: Die Kapazität des Haupt- und des Sekundärspeichers reichen zur Aufnahme des Betriebssystemkerns, des Codes und der Daten eines aktivierten Prozeßsystems bzw. zur Abspeicherung einer Prozeßsystem-Kopie und zur Aufzeichnung der zu einer Berechnung festgehaltenen Historie existierender Prozeßsystem-Zustände aus.

Annahme 2.2.1.3/2: Ab dem Zeitpunkt des Beginns einer Berechnung, d.h. der Aktivierung eines Prozeßsystems, bis zu ihrem ordnungsgemäßen Abschluß, können am Sekundärspeicher zu diskreten Zeitpunkten lokale Zustände aktiver Prozesse aufgezeichnet werden und die Werte globaler Variablen bei Wertänderung kontinuierlich fortgeschrieben werden.

2.2.1.4. Verhalten im Fehlerfall

Das Verhalten eines realen Rechensystems läßt sich in zwei Klassen von Verhaltensmustern unterteilen, [Mos81]: in *erwartetes* (akzeptables) und in *unerwartetes* (nicht akzeptables) Verhalten.
Im Falle vom erwarteten Verhalten kann weiter unterschieden werden in *korrektes* und in *fehlerhaftes*, aber *tolerierbares* Verhalten. Unerwartetes Verhalten bezeichnet ein fehlerhaftes Verhalten, das *nicht tolerierbar* ist.
Als *fehlerhaft* wird ein Verhalten dann bezeichnet, wenn es vom erwünschten, spezifizierten Verhalten abweicht, andernfalls wird es als korrekt bezeichnet.

Ob ein fehlerhaftes Verhalten eines Rechensystems tolerierbar ist, hängt von der Art der Systembenutzung und der Reaktion des Rechensystems auf Fehler ab. Im vorliegenden Fall steht die Nutzung durch ein Prozeßsystem im Vordergrund.
Aus Prozeßsystemsicht ist ein fehlerhaftes Verhalten eines Rechensystems dann tolerierbar, wenn es als fehlerhaft erkennbar ist, sowohl seine Ursachen als auch seine Auswirkungen eingrenzbar sind, diese beseitigt werden können, und ein eventuell entstandener fehlerhafter Systemzustand (Prozeß- und Rechensystem-Zustand) korrigierbar ist.

Die im Rahmen dieser Arbeit vorgestellten Mechanismen zur Fehlerbehandlung in Prozeßsystemen setzen ein fehlerfrei funktionierendes Rechensystem zur Erreichung ihrer Wirksamkeit voraus. Unter Einhaltung bestimmter Anforderungen an das Verhalten eines Rechensystems im Fehlerfall ermöglichen sie jedoch prinzipiell auch die Tolerierung auftretender Rechensystemfehler. Als dafür hinreichend wird in diesem Zusammenhang beispielhaft zunächst die folgende Annahme getroffen:

Annahme 2.2.1.4/1: In den betrachteten, Prozeßsystemen unterlagerten Rechensystemen treten ausschließlich Fehler auf, die von aktiven Berechnungen toleriert werden können.

Um diese Annahme im Rahmen des vorgestellten Rechensystem-Modells als erfüllt betrachten zu können, wird im folgenden vorausgesetzt, daß jeder im Hauptspeichermodul, am Sekundärspeicher, in der Rechen- und Steuereinheit, auf den Kanal- und Busverbindungen oder in der Basissoftware auftretende Fehler unmittelbar erkannt wird und, soweit er nicht für aktive Prozeßsystemabläufe transparent behebbar ist, zum Abbruch aktiver Berechnungen durch sofortigen Rechensystem-Stop führt. Diese Überlegungen lassen sich in der folgenden Annahme zusammenfassen:

Annahme 2.2.1.4/2: Vom unterlagerten Rechensystem nicht prozeßsystemtransparent tolerierbare Fehler führen zum Abbruch laufender Berechnungen durch sofortigen Rechensystem-Stop

Im Kontext des skizzierten Modells bedeutet dies, daß zum einen aktive Berechnungen nicht durch Fehlverhalten eines Rechensystems beeinflußt werden können; zum anderen können Fehler eines Rechensystems unter Annahme 2.2.1.4/2 durch ein Prozeßsystem genau dann toleriert werden, wenn letzteres die ordnungsgemäße Fortsetzung abgebrochener aktiver Berechnungen über Rechensystem-Stops hinweg unterstützt.

Mit einem Rechensystem-Stop ist in realen Systemen häufig der Verlust des Hauptspeicherinhaltes verbunden und damit in der Regel auch der Verlust der aktuellen Systemzustände aktivierter, d.h. in Ausführung befindlicher Prozeßsysteme. Somit ist ein Fortsetzen abgebrochener Berechnungen, unter der Voraussetzung, daß alle Fehlerursachen beseitigt worden sind, nur über einen Rechensystem-Neustart möglich, mittels dessen ein Rechensystem in einen definierten, fehlerfreien Anfangszustand versetzt wird, von dem aus ein Wiederaufsetzen abgebrochener aktiver Abläufe versucht werden kann.

Die Fortsetzung einer aktiven Berechnung nach erfolgreichem Wiederanlauf durch Rechensystem-Neustart setzt die Existenz einer vor Rechensystemfehlern geschützten Prozeßsystem-Kopie voraus. Soll des weiteren ein Fortsetzen abgebrochener Abläufe nicht ausschließlich darin bestehen eine Berechnung vom Anfangszustand ausgehend komplett neu zu beginnen, so muß die Rekonstruierbarkeit eines möglichst unmittelbar vor dem Abbruch einer Berechnung existierenden fehlerfreien Zwischenzustandes gewährleistet sein. Diese Überlegung führt schließlich bzgl. des vorgestellten Rechensystem-Modells zu den folgenden beiden Annahmen:

Annahme 2.2.1.4/3: Die am Sekundärspeicher eines Rechensystems gehaltenen Informationen können durch Fehler in einzelnen Systemkomponenten nicht zerstört und auch nicht verfälscht werden, oder anders formuliert: Jede am Sekundärspeicher gesicherte Information ist zu jedem Zeitpunkt fehlerfrei rekonstruierbar.

Annahme 2.2.1.4/4: Die für den (lesenden/schreibenden) Zugriff auf den Sekundärspeicher vom Betriebssystemkern bereitgestellten System-Aufrufe arbeiten gemäß ihrer Spezifikation. Zusätzlich besitzen sie die Eigenschaft atomar zu sein, d.h. sie werden vollständig und korrekt ausgeführt; im Falle eines Abbruchs werden die durch sie bedingten Änderungen nicht sichtbar.

Auf welche Weise die Annahmen, zumindest mit einer hohen Wahrscheinlichkeit, in realen Systemen erfüllt werden können wird an dieser Stelle nicht näher diskutiert. Vorschläge hierzu findet man u.a. in [LS81], [Mos81], [Ree78] und in [SM180].

2.2.2. Vorausgesetztes Zustandssicherungskonzept

Wie im vorangehenden Abschnitt erläutert, wird ein erfolgreiches Wiederaufsetzen unterbrochener, aktiver Berechnungen im vorliegenden Fehlerbehandlungs-Modell mittels des Fortschreibens lokaler Prozeßsystem-Zustände und der Aufzeichnung der Werte globaler Variablen bei Wertänderung ermöglicht (vgl. Annahme 2.2.1.3/2). Neben detaillierten Angaben zu den vorausgesetzten Mechanismen zur Aufzeichnung und Verwaltung von Sicherungsinformation am Sekundärspeicher werden auch die zugrundegelegten Speicherungsstrukturen beschrieben. Abschließend wird das skizzierte Prozeßzustandssicherungskonzept anhand eines ausführlichen Beispiels veranschaulicht und näher erläutert.

2.2.2.1. Historie prozeßlokaler Wiederaufsetzpunkte

Die Sicherung des prozeßlokalen Zustandes, d.h. der Werte lokaler Variablen, eines beliebigen Prozesses P_i innerhalb eines Prozeßsystems P erfolgt im gewählten Modell mittels des Betriebssystemkernaufrufes $up(P_i)$ (eine genaue Definition der Operation $up(\ )$ folgt) nach dem folgenden Schema: Zu jedem aktivierten Prozeßsystem P wird vom Betriebssystemkern am Sekundärspeicher ein Bereich, der als Zustands-Historie bezeichnet wird, verwaltet. Je Prozeß P_i wird in diesem Bereich eine Liste $h(P_i)$ aller gesicherten prozeßlokalen Zustände des Prozesses P_i, verkettet in der zeitlichen Reihenfolge ihres Anlegens, geführt. Der Aufruf $up(P_i)$ bewirkt eine Erweiterung von $h(P_i)$ um ein neues Element, das den momentan aktuellen lokalen Prozeßzustand des Prozesses P_i beschreibt.

Definition 2.2.2.1/1: Sei $P=(A,D,val_P(\alpha))$ ein aktiviertes Prozeßsystem von n Prozessen $P_1, \ldots, P_n$, das sich im Zustand $val_P(\alpha)$ befindet:

(1) Mittels $val_P(\ ^\bullet)$ wird im folgenden der aktuelle Prozeßsystem-Zustand von P bezeichnet, d.h.:

$$val_P(\ ^\bullet) ::= val_P(\alpha).$$

(2) Mit $val_P(\ ^\bullet|P_i)$ wird derjenige Zustand, der durch Projektion des aktuellen Systemzustandes auf die Werte der lokalen Variablen des Prozesses P_i, $i \in \{1,\ldots,n\}$, entsteht, bezeichnet. Das bedeutet:

$$v[val_P(\ ^\bullet|P_i)] ::= \begin{cases} v[val_P(\ ^\bullet)] & v \in V_i \ \&\ l(v)=\textbf{true} \\ & \textit{falls} \\ \textbf{nil} & v \notin V_i \ \|\ l(v)=\textbf{false} \end{cases}$$

wobei das Kürzel **nil** diejenigen Stellen von $val_P(\bullet)$ kennzeichnet, welche bei der Projektion nicht berücksichtigt werden.

(3) Durch $val_P(\alpha|P_i)$ wird analog zu (2) der Zustand beschrieben, welcher durch Projektion des Systemzustandes auf Werte der lokalen Variablen des Prozesses P_i, $i \in \{1,...,n\}$, entsteht.

Die Operation $up(P_i)$, mit deren Hilfe ein lokaler Wiederaufsetzpunkt zu Prozeß P_i am Sekundärspeicher gesichert werden kann, läßt sich unter Bezugnahme auf Definition 2.2.2.1/1 wie folgt beschreiben.

Definition 2.2.2.1/2: Sei P ein aktiviertes Prozeßsystem und $h(P_i)$ die Liste aller gesicherten lokalen Zustände des Prozesses P_i, dann läßt sich $up(P_i)$ definieren als:

$$up(P_i) ::= (h(P_i) := h(P_i) \circ val_P(\bullet|P_i)),$$

hierbei liefert $h(P_i) \circ val_P(\bullet|P_i)$ die durch Anfügen von $val_P(\bullet|P_i)$ an $h(P_i)$ entstehende Liste (*Konkatenation* von $val_P(\bullet|P_i)$ an $h(P_i)$) und der Anfangswert von $h(P_i)$ ist definiert als $val_P(w|P_i)$.

Nach ihrem Urheber lassen sich analog der in [Kan81] eingeführten Differenzierung *prozeßsysteminitiierte* und *betriebssysteminitiierte* Operationen zum Anlegen prozeßlokaler Wiederaufsetzpunkte unterscheiden. Als prozeßsysteminitiierte Sicherungsaufrufe können alle Aktionen $\alpha \in A$ eines Prozeßsystems $P=(A,D,w)$ bezeichnet werden, in deren Operationsteil die Anweisung $up(...)$ vorkommt. Betriebssysteminitiierte Anweisungen zum Einrichten von Wiederaufsetzpunkten sind hingegen nicht explizit sichtbar; abhängig von der durch ein Betriebssystem verfolgten Prozeßablaufsicherungs-Strategie werden sie prozeßsystemtransparent gesetzt.

Im weiteren wird mit Zustandssicherungsaktionen $\alpha_{i,j}$ die folgende Struktur verbunden und im Zusammenhang mit durch eine Aktion $\alpha_{i,j}$ prozeßsysteminitiierten Wiederaufsetzpunkten $up(P_i)$ wird angenommen, daß der in $up(P_i)$ gesicherte Befehlszählerzustand auf die Nachfolgeraktion von $\alpha_{i,j}$ im Prozeß P_i innerhalb der gerade aktuellen Berechnung zeigt.

Definition 2.2.2.1/3: Sei $P=(A,D,w)$ ein Prozeßsystem. Eine Aktion $\alpha_{i,j} \in A$ wird als eine den lokalen Prozeßsystem-Zustand von P_i *sichernde Aktion* bezeichnet genau dann, wenn gilt:

$$\alpha_{i,j}: \textit{when } (bz_i = adr(\alpha_{i,j})) \textit{ do } up(P_i);\ bz_i := adr(\alpha_{i,j})';\ \textit{od};$$

Kurzschreibweise $up(a_{i,j})$= **true** <=> $a_{i,j}$ ist eine den Prozeßsystem-Zustand von P_i sichernde Aktion.

Dem in dieser Arbeit verfolgten Fehlerbehandlungskonzept für Prozeßsysteme liegt bzgl der Sicherung prozeßlokaler Variablenwerte die folgende Annahme zugrunde.

Annahme 2.2.2.1/1: Es wird davon ausgegangen, daß prozeßlokale Wiederaufsetzpunkte zu diskreten Zeitpunkten sowohl prozeßsysteminitiiert als auch abhängig von der evtl. vom Betriebssystem verfolgten Dominoeffektverhinderungs-Strategie betriebssysteminitiiert angelegt werden können. Eine lückenlose Aufzeichnung aller Änderungen lokaler Variablenwerte wird nicht vorausgesetzt.

2.2.2.2. Historie globaler Variablenwerte

Die Sicherung prozeßlokaler Variablenwerte, wie in Abschnitt 2.2.2.1 beschrieben, reicht allein im allgemeinen nicht aus, um im Fehlerfall den Aufwand zur Wiederherstellung eines konsistenten Zustandes in Grenzen zu halten, und auf diese Weise der Gefahr eines Dominoeffektes zu begegnen. Aus diesem Grunde wird im vorliegenden Modell von der folgenden Annahme ausgegangen:

Annahme 2.2.2.2/1: Zusätzlich zu den im gewählten Rechensystem-Modell mittels prozeßlokaler Wiederaufsetzpunkte gesicherten Werten lokaler Variablen werden in dem Bereich Zustands-Historie, der, wie bereits erwähnt, zu jedem aktivierten Prozeßsystem auf dem Sekundärspeicher geführt wird, zu jeder globalen Variablen $gv \in V_P$ eines Prozeßsystems $P=(A,D,w)$ alle Wertänderungen in zeitlicher Reihenfolge in einer gv eindeutig zugeordneten Liste $h(gv)$ lückenlos aufgezeichnet.

Annahme 2.2.2.2/2: Das Eintragen von Elementen in die zu einer globalen Variablen gv geführte Historie $h(gv)$ bei Wertänderung von gv erfolgt implizit durch Funktionen des Betriebssystemkerns, und da die Aufzeichnung der Werte nach Annahme 2.2.2.2/1 lückenlos erfolgt, werden Prozeßsystemen vom Betriebssystemkern keinerlei Operationen zur expliziten Sicherung globaler Variablenwerte zur Verfügung gestellt.

Annahme 2.2.2.2/3: Bzgl des Sicherungszeitpunktes des Wertes einer globalen Variablen gv gilt, daß eine Sicherung des aktuellen Wertes gleichzeitig und atomar mit jeder Änderung von gv erfolgt.

2.2.2.3. Zugrundegelegte Speicherungsstrukturen

Die Existenz der Listen $h(P_i)$ und $h(gv)$ zu jedem Prozeß P_i bzw. jeder globalen Variablen $gv \subset V_P$ eines Prozeßsystems $P=(A,D,w)$ ist eine der Voraussetzungen, um im Fehlerfall einen konsistenten Zustand zu einer abgebrochenen Berechnung wiederherstellen zu können und eine Fortsetzung mit begrenzbarem Aufwand zu ermöglichen. Ebenso wichtig ist aber auch das Vorhandensein einer Vorschrift, nach der die einzelnen Elemente der existierenden Historien des Typs $h(P_i)$ zu Elementen der Listen $h(gv)$ derart zugeordnet werden können, daß von der Zustands-Historie ausgehend ableitbar ist, welcher Prozeß jeweils für die Änderung eines Wertes einer globalen Variablen verantwortlich ist, und welche Prozesse bestimmte Werte globaler Variablen gelesen haben.
Durch folgende Verkettung einzelner Elemente der Listen des Typs $h(P_i)$ mit denjenigen des Typs $h(gv)$ kann die Erfüllung der letztgenannten Forderung gewährleistet werden:

Annahme 2.2.2.3/1: Sei $h(P_i)[j]$ das j'te Element der Liste $h(P_i)$ und $h(gv)[k]$ sei analog definiert. Zu jedem Element $h(P_i)[j]$ existiert eine *Liste globaler Schreibreferenzen* (*LgSR*) und eine *Liste globaler Lesereferenzen* (*LgLR*). In *LgSR* wird ein Verweis auf ein Element $h(gv)[k]$ genau dann eingetragen, wenn $h(gv)[k]$ aufgrund einer Änderung von gv durch Prozeß P_i angelegt wurde und $h(gv)[k]$ in der zeitlichen Relation

$$t(h(P_i)[j]) < t(h(gv)[k]) < t(h(P_i)[j+1])$$

zu $h(P_i)[j]$ steht. In *LgLR* wird genau dann ein Verweis auf $h(gv)[k]$ eingefügt, wenn innerhalb des Intervalls $[t(h(P_i)[j]),t(h(P_i)[j+1])]$ von P_i aus auf den in $h(gv)[k]$ gesicherten Wert lesend zugegriffen wurde.
Mit $t(h(P_i)[j])$, $t(h(P_i)[j+1])$ und $t(h(gv)[k])$ wird der Zeitpunkt bezeichnet, zu dem $h(P_i)$ bzw. $h(gv)$ um das Element $h(P_i)[j]$, $h(P_i)[j+1]$ bzw. $h(gv)[k]$ erweitert wurden.

Annahme 2.2.2.3/2: Zu jedem Element $h(gv)[k]$ wird eine (einelementige Liste) *globale Schreibreferenz* (*gSR*) und eine *Liste globaler Lesereferenzen* (*LgLR*) geführt. Dabei bezieht sich *gSR* genau dann auf ein Element $h(P_i)[j]$, wenn $h(gv)[k]$ aufgrund einer Änderung von gv durch Prozeß P_i angelegt wurde und $h(gv)[k]$ in der folgenden zeitlichen Relation zu $h(P_i)[j]$ steht:

$$t(h(P_i)[j]) < t(h(gv)[k]) < t(h(P_i)[j+1]).$$

Die Liste *LgLR* enthält genau dann einen Verweis auf $h(P_i)[j]$, wenn innerhalb des Intervalls $[t(h(P_i)[j]),t(h(P_i)[j+1])]$ von P_i aus auf $gv[k]$ lesend zugegriffen wurde.

In einer der Programmiersprache C ähnlichen Notation formuliert, in der das Symbol "↑" als Abkürzung für *Zeiger auf* und das Symbol "[]" zur Vereinbarung *variabel langer Felder* Verwendung finden, wird im weiteren, unter Berücksichtigung der Annahmen 2.2.2.3/1 und 2, für Elemente des Typs $h(P_i)$ die folgende Datenstruktur zur Abspeicherung am Sekundärspeicher vorgesehen:

```
struct element_of_h(P_i) {
        local_variable bz_i;
        local_variable lv_i[];
        set_of struct ↑element_of_h(gv) LgSR;
        set_of struct ↑element_of_h(gv) LgLR;
      };
```

und für Elemente des Typs $h(gv)$ die Struktur:

```
struct element_of_h(gv) {
        global_variable gv;
        struct ↑element_of_h(P_i) gSR;
        set_of struct ↑element_of_h(P_i) LgLR;
      };
```

Historien $h(P_j)$ und $h(gv_k)$ werden am Sekundärspeicher mittels Felder variabler Länge (Felder ohne feste Dimensionsgrenzen) repräsentiert:

```
struct element_of_h(P_i) h(P_j)[];
struct element_of_h(gv) h(gv_k)[];
```

2.2.2.4. Veranschaulichung des Zustandssicherungskonzeptes

Im folgenden wird das skizzierte Sicherungskonzept anhand eines Beispiel-Prozeßsystems (siehe Bild 2.2.2.4/1), dem das im Bild 2.1.1.4/1 dargestellte Prozeßsystem entspricht, ergänzt um je eine Wiederaufsetzpunktaktion pro Prozeß zur Sicherung, erläutert.

Die Bilderfolge 2.2.2.4/2 bis 10 zeigt die Zustandsänderungen des Beispiel-Prozeßsystems P bezogen auf den aktuellen Zustand (hauptspeicherresident) und die zu P geführte Zustands-Historie (sekundärspeicherresident) während der Berechnung $\alpha ::= a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4}$ nach jedem einzelnen Berechnungsschritt, beginnend mit dem Anfangszustand $val_P(\lambda)$.
Bei der Darstellung der Historien des Typs $h(P_i)$ und $h(gv)$ wird davon ausgegangen,

Anfangswerte: $gv_1 := 0;$ $gv_2 := 0;$ $gv_3 := 3;$
$bz_1 := 1;$ $lv_{1,1} := 0;$ $bz_2 := 1;$ $lv_{2,1} := 0;$

Prozeß P_1:

$a_{1,1}$: *when* $(bz_1 = adr(a_{1,1}))$ *do*
$bz_1 := (if\ (lv_{1\,1} \geq gv_3)\ then\ adr(a_{1,5})\ else\ adr(a_{1,2}););$
od;

$a_{1,2}$: *when* $((bz_1 = adr(a_{1,2}))\&(gv_1 \leq 0))$ *do*
$gv_2 := gv_2 + 1;$
$gv_1 := gv_1 + 1;$
$bz_1 := adr(a_{1,3});$
od;

$a_{1,3}$: *when* $(bz_1 = adr(a_{1,3}))$ *do*
$lv_{1,1} := lv_{1,1} + 1;$
$bz_1 := adr(a_{1,4});$
od;

$a_{1,4}$: *when* $(bz_1 = adr(a_{1,4}))$ *do*
$h(P_1) := (h(P_1) \circ val_P(* \mid P_1));$
$bz_1 := adr(a_{1,1});$
od;

$a_{1,5}$: *when* $(bz_1 = adr(a_{1,5}))$ *do*
exit(P_1);
od;

Prozeß P_2:

$a_{2,1}$: *when* $(bz_2 = adr(a_{2,1}))$*do*
$bz_2 := (if\ (lv_{2,1} \geq gv_3)\ then\ adr(a_{2,5})\ else\ adr(a_{2,2}););$
od;

$a_{2,2}$: *when* $((bz_2 = adr(a_{2,2}))\&(gv_1 \geq 1))$ *do*
$gv_2 := gv_2 * gv_2;$
$gv_1 := gv_1 - 1;$
$bz_2 := adr(a_{2,3});$
od;

$a_{2,3}$: *when* $(bz_2 = adr(a_{2,3}))$ *do*
$lv_{2,1} := lv_{2,1} + 1;$
$bz_2 := adr(a_{2,4});$
od;

$a_{2,4}$: *when* $(bz_2 = adr(a_{2,4}))$ *do*
$h(P_2) := (h(P_2) \circ val_P(* \mid P_2));$
$bz_2 := adr(a_{2,1});$
od;

$a_{2,5}$: *when* $(bz_2 = adr(a_{2,5}))$ *do*
exit(P_2);
od;

Bild 2.2.2.4/1: Beispiel-Prozeßsystem P mit Wiederaufsetzpunkt-Sicherung

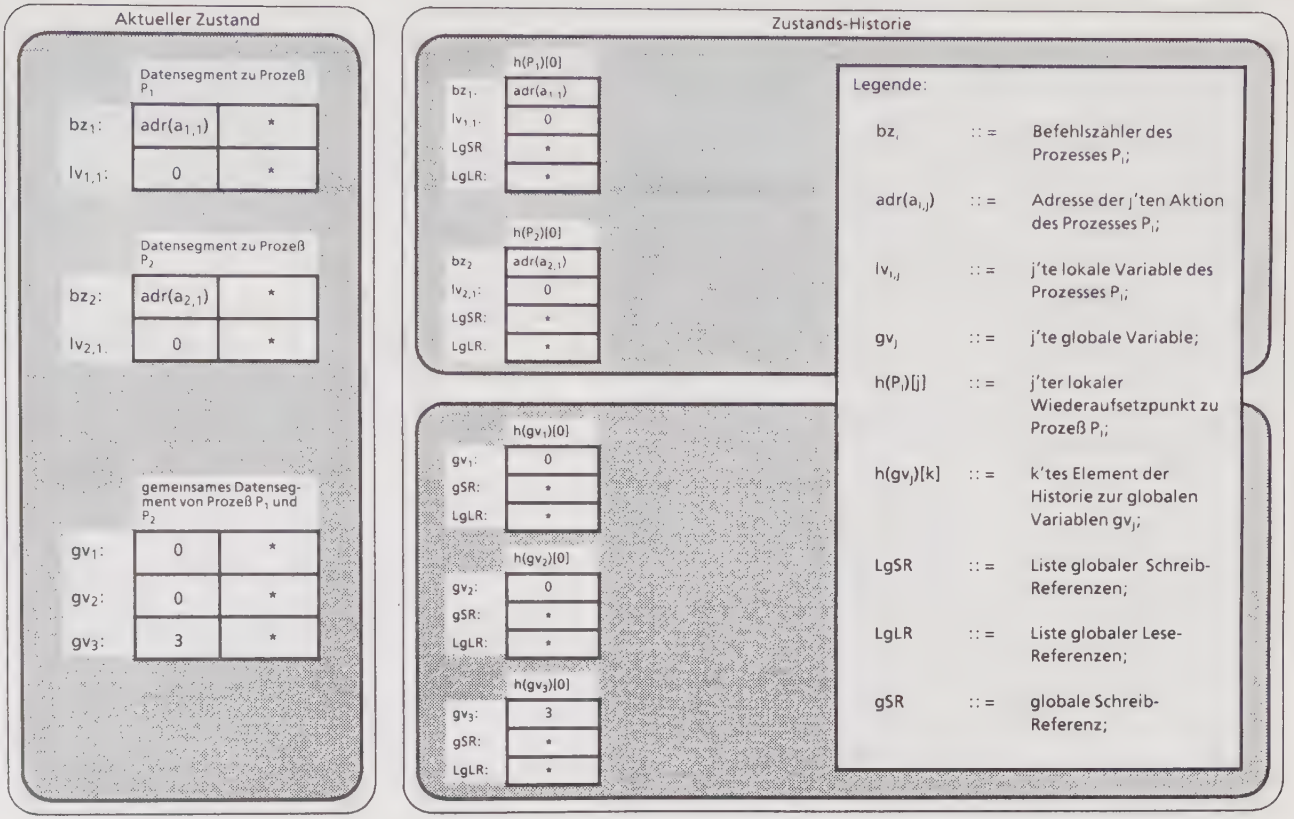


Bild 2.2.2.4/2: Prozeßsystem-Zustand $val_P(\lambda)$

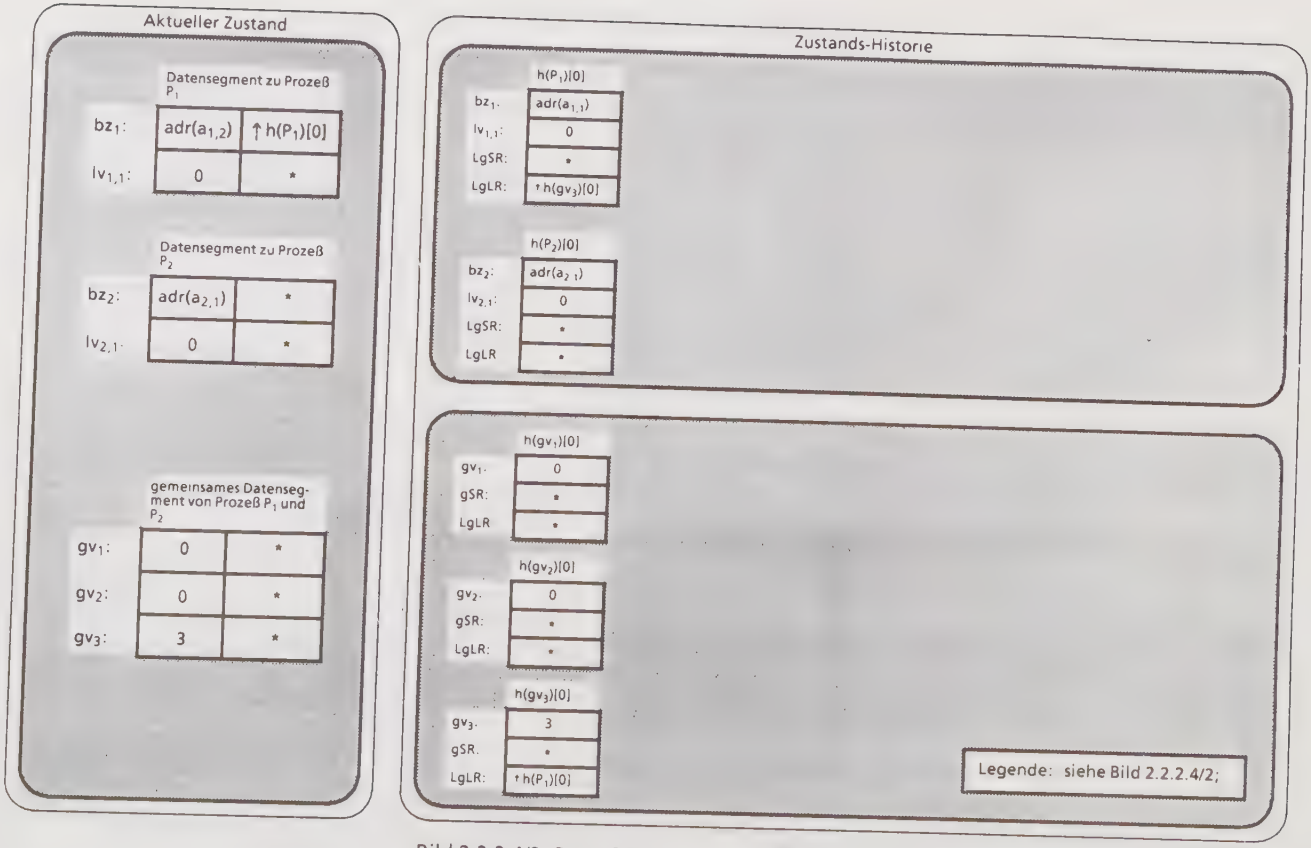


Bild 2.2.2.4/3: Prozeßsystem-Zustand $val_P(a_{1,1})$

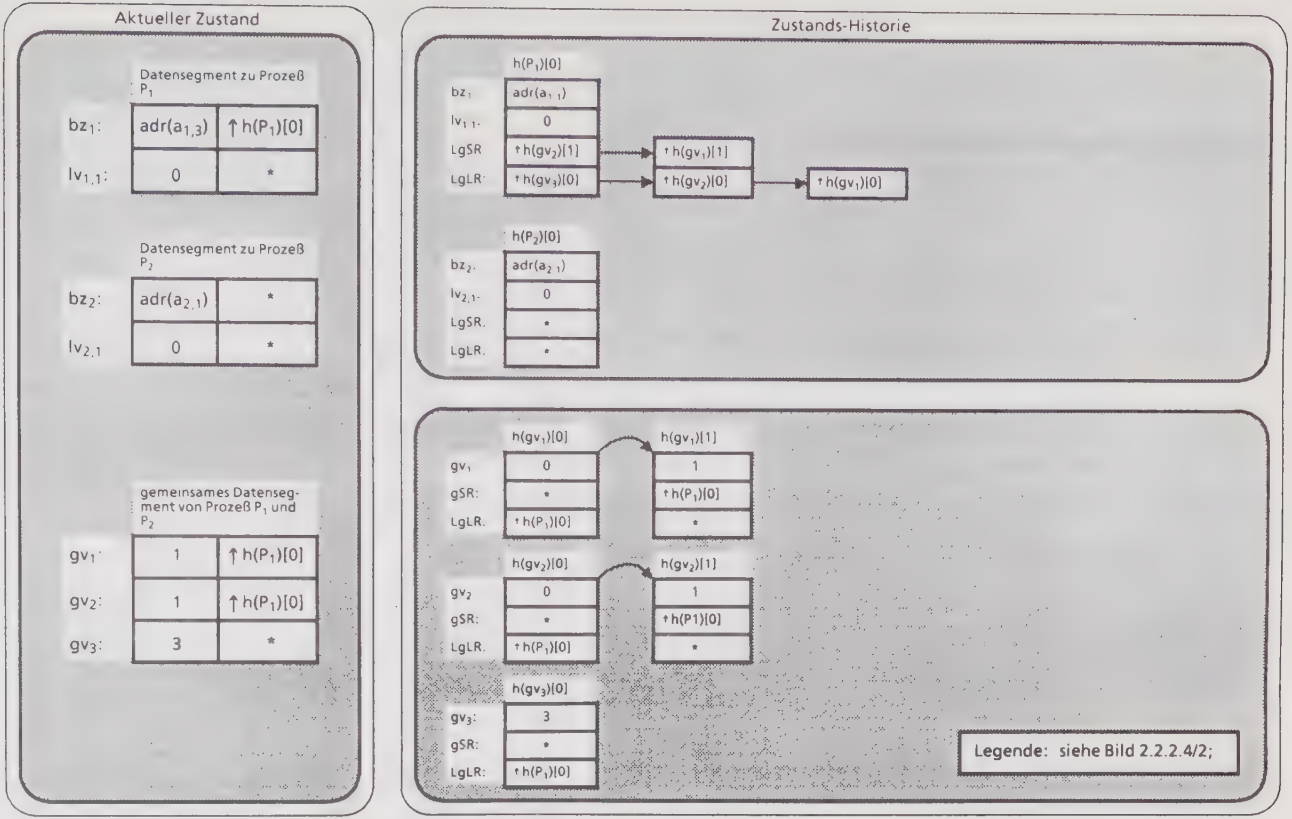


Bild 2.2.2.4/4: Prozeßsystem-Zustand *valp*($a_{1,1}a_{1,2}$)

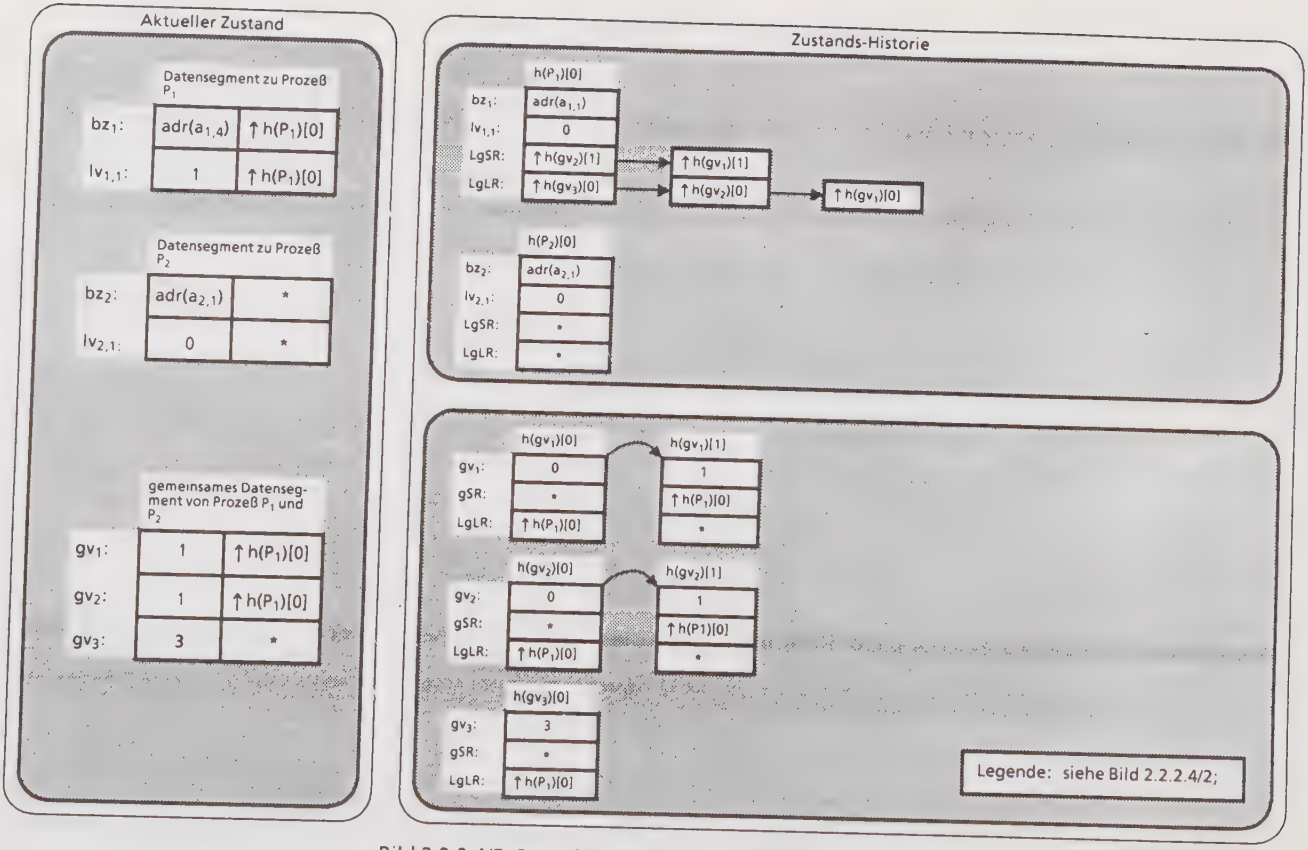


Bild 2.2.2.4/5: Prozeßsystem-Zustand $val_P(a_{1,1}a_{1,2}a_{1,3})$

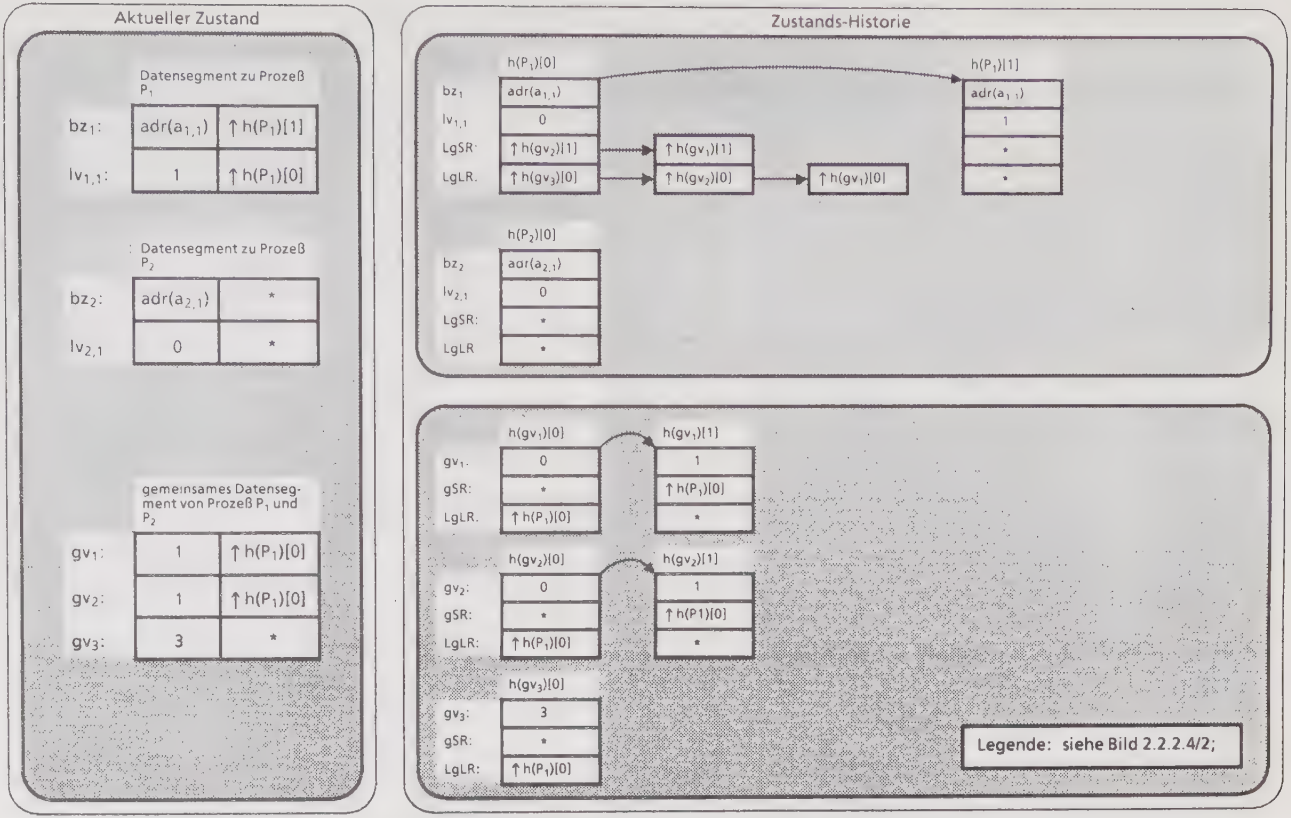


Bild 2.2.2.4/6: Prozeßsystem-Zustand *valp*($a_{1,1}a_{1,2}a_{1,3}a_{1,4}$)

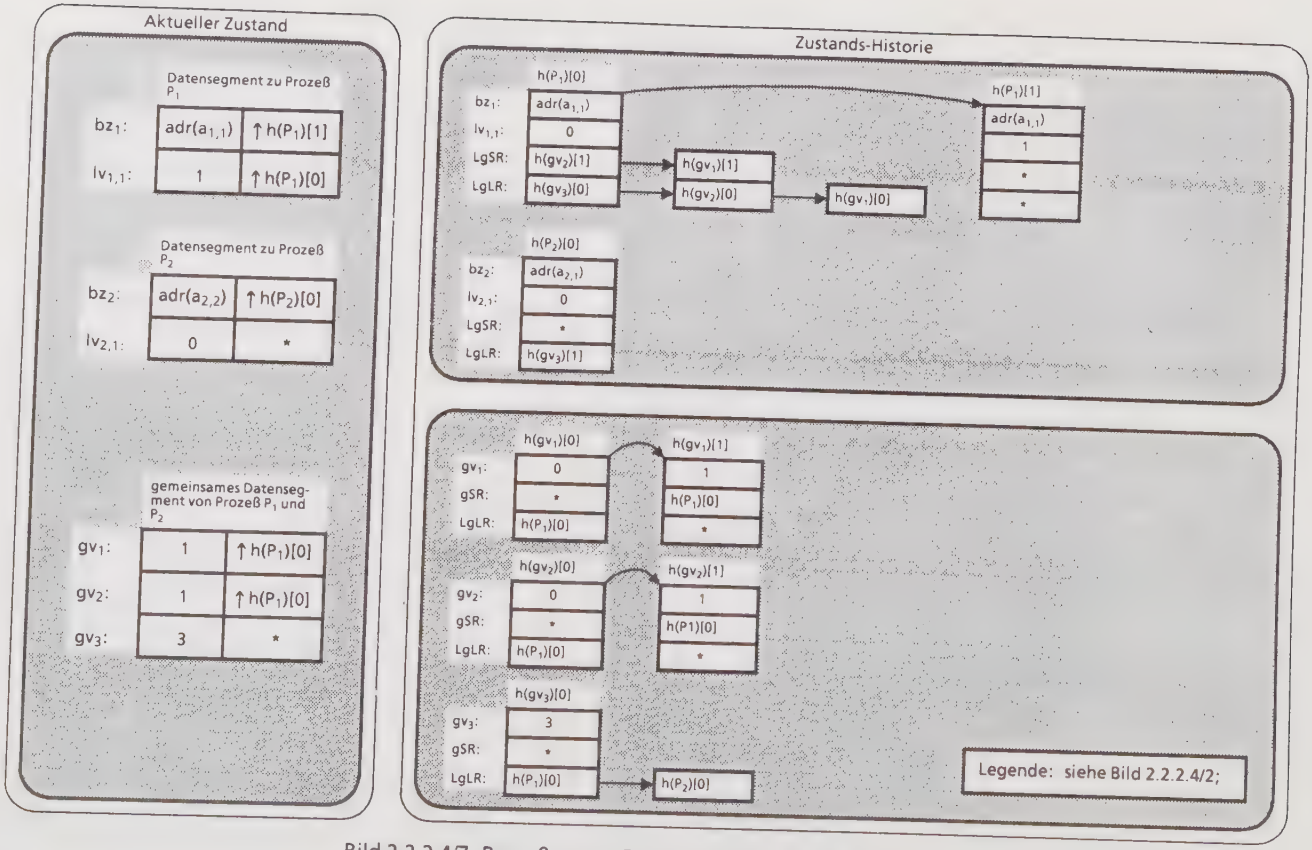


Bild 2.2.2.4/7: Prozeßsystem-Zustand $valp(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1})$

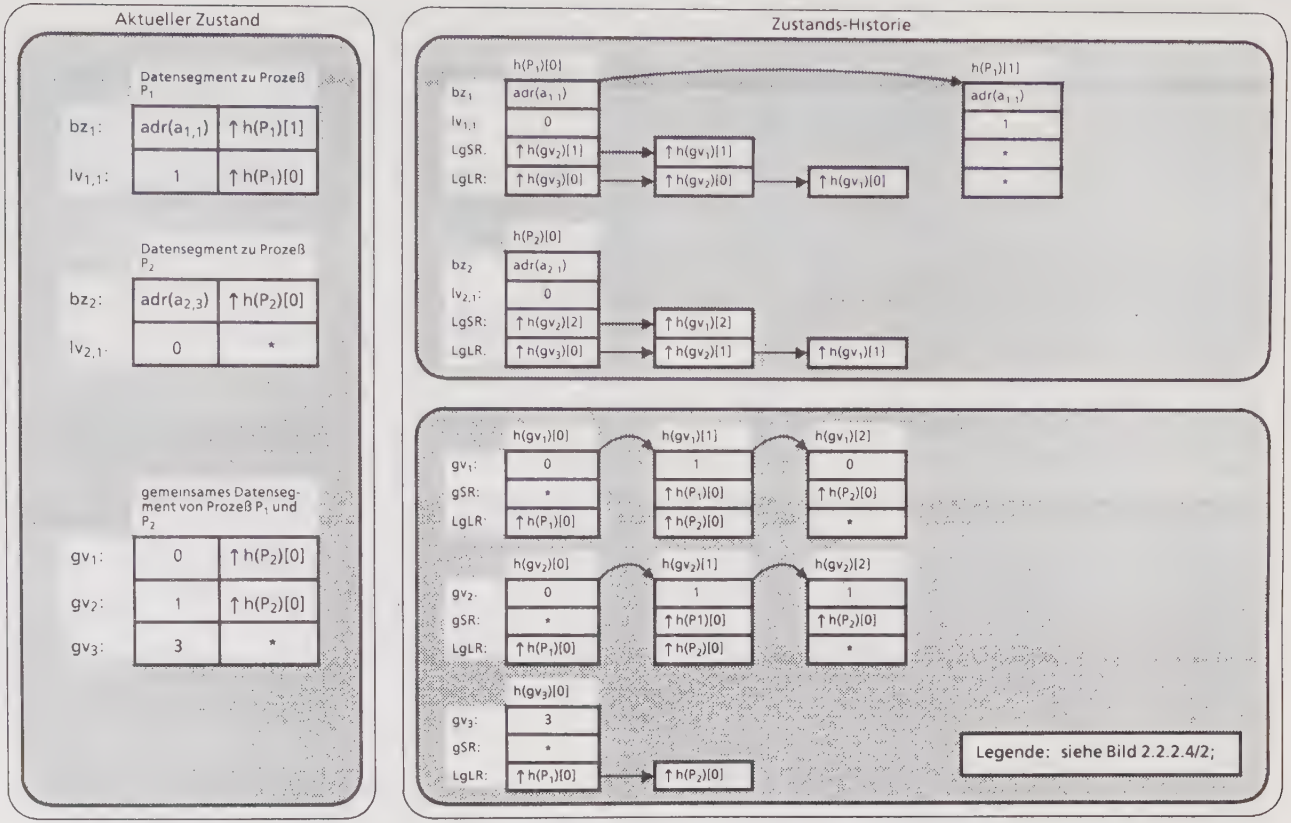


Bild 2.2.2.4/8: Prozeßsystem-Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2})$

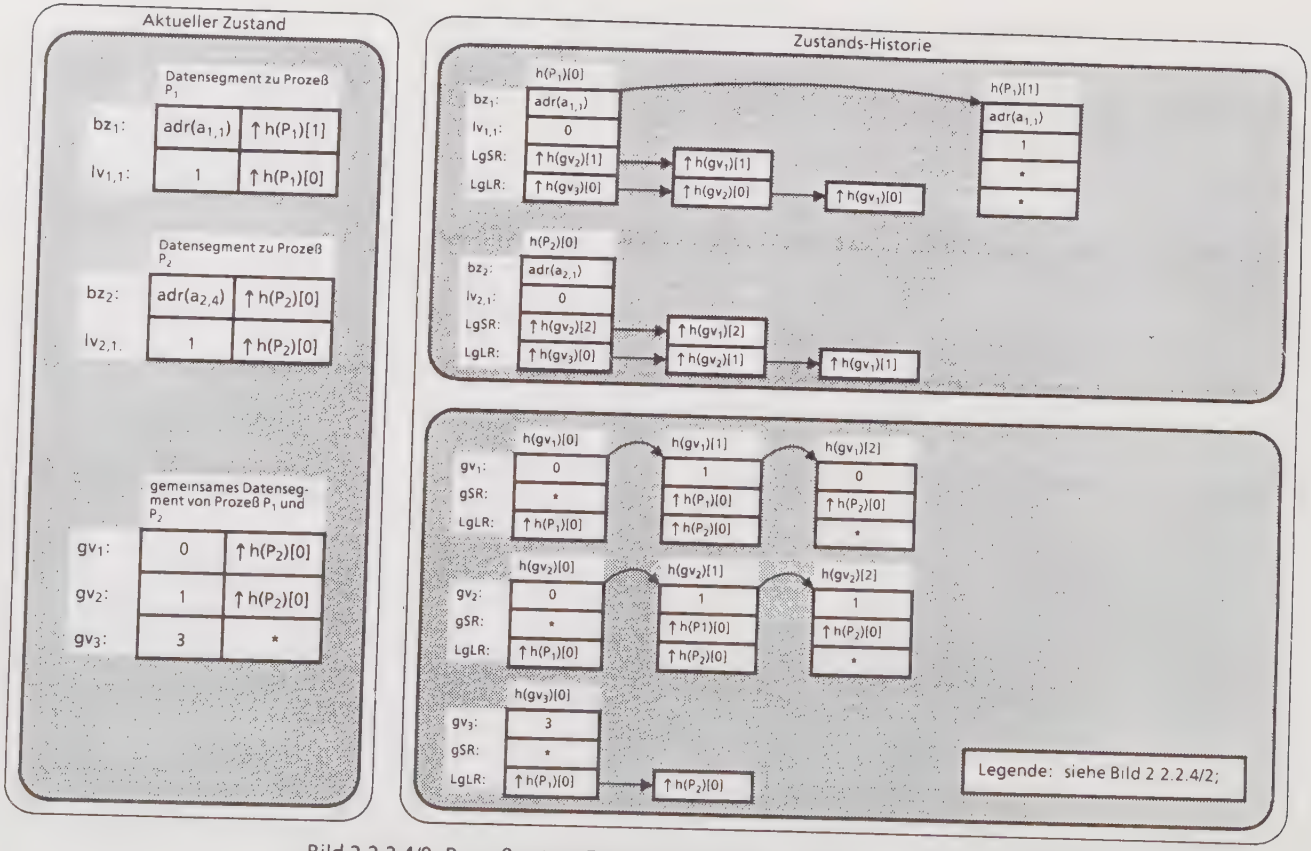


Bild 2.2.2.4/9: Prozeßsystem-Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3})$

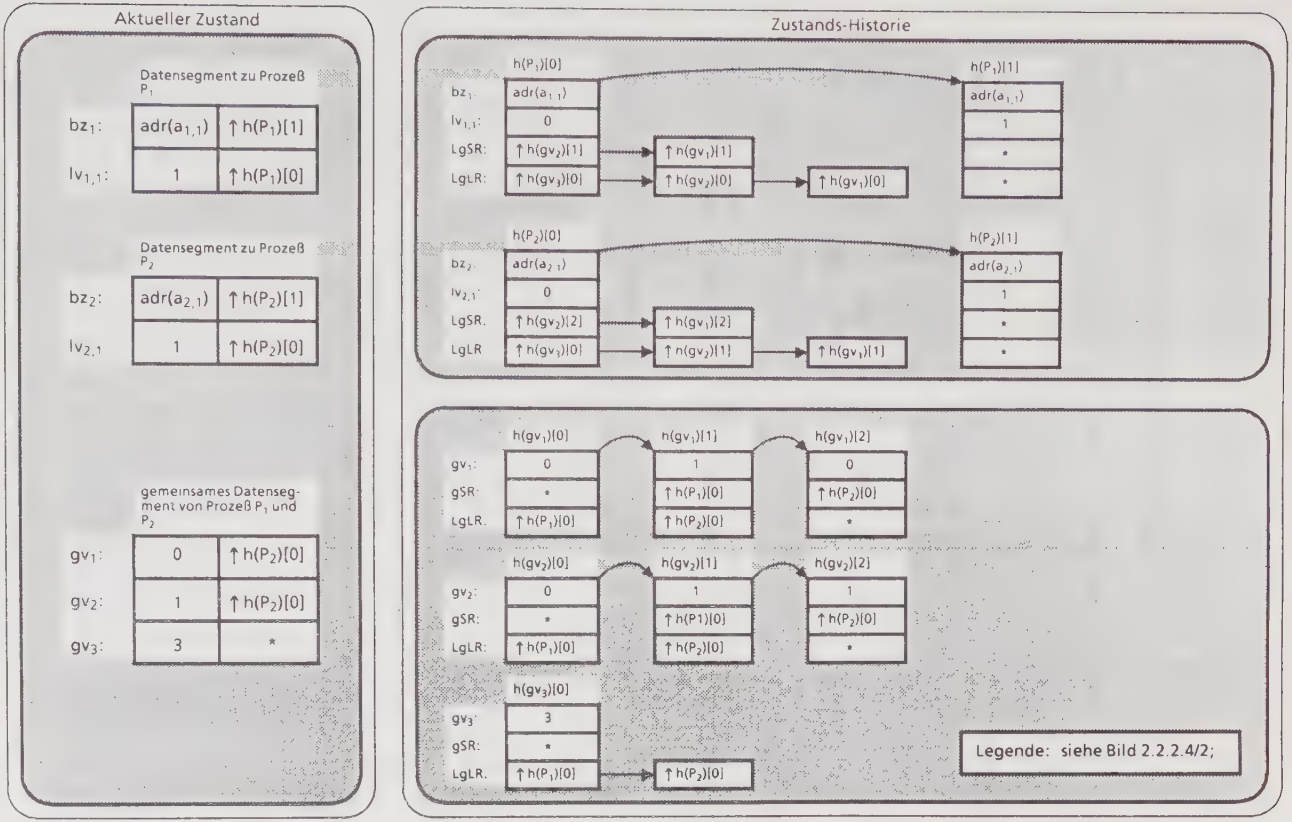


Bild 2.2.2.4/10: Prozeßsystem-Zustand *valp*($a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4}$)

daß die als Felder variabler Länge deklarierten Speicherungsstrukturen mittels verketteter Listen realisiert werden und die Elemente $h(P_i)[0]$ und $h(gv_j)[0]$, $i \in \{1,2\}$, $j \in \{1,2,3\}$, vor dem Berechnungsstart implizit durch den Betriebssystemkern angelegt werden.

Erläuterungen zur Notation der Bilder 2.2.2.4/2 bis 10: Leere Verweise (*Nullzeiger*) werden in den Bildern durch das Kürzel "*" repräsentiert. Des weiteren wird, zusätzlich zu ihrem Wert, im Hauptspeicher pro Variable v ein Verweis auf denjenigen Wiederaufsetzpunkt geführt, zu welchem der v verändernde Prozeß zurückgesetzt werden mußte, falls sich der aktuelle Wert von v als unzulässig/korrekturbedürftig erweisen sollte. Dieser Einstiegspunkt zu einer Fehlerbehandlung ist dann von Bedeutung, wenn eine Fehlerkorrektur von einem aktuellen Wert einer Variablen ausgehend erfolgen soll. Prinzipiell ist das Führen derartiger Information aber nur unter Effizienzgesichtspunkten betrachtet als erforderlich anzusehen, denn die im Fehlerfall benötigten Einstiegspunkte können, eine sorgfältige Speicherung vorausgesetzt, auch von der zu einem aktivierten Prozeß existierenden Zustands-Historie aus rekonstruiert werden.
In diesem Zusammenhang ist zu bemerken, daß ein "*"-Eintrag neben einem Variablenwert des aktuellen Zustandes diesen als Initialisierungswert kennzeichnet.

Weitere Erläuterungen zu den Bildern 2.2.2.4/2-10 enthält die in Bild 2.2.2.4/2 zu den verwendeten Abkürzungen zusammengestellte Legende.

2.2.3. Zugrundegelegtes Wiederanlaufverfahren

Auf dem in Abschnitt 2.2.2 präzisierten Prozeßsystem-Zustandssicherungskonzept lassen sich verschiedene Algorithmen, die nach einem Berechnungsabbruch die Wiederherstellung eines konsistenten Zustandes ermöglichen, aufbauen. Nach einer Erläuterung prinzipiell zu beachtender und zu unterscheidender Fehlerfälle, wird das den folgenden Diskussionen zugrundegelegte Wiederaufsetzverfahren skizziert und anhand von Beispielen seine Funktionsweise aufgezeigt. Schließlich werden die einzelnen im Fehlerfall zur Wiederherstellung eines konsistenten Zustandes zu durchlaufenden Rücksetzschritte mittels eines Petri-Netzes anschaulich dargestellt.

2.2.3.1. Zu beachtende Fehlerfälle

Das in dieser Arbeit zugrundegelegte Fehlerbehandlungs-Modell beruht auf der Unterscheidung von zwei Fehlertypen *Rechensystemfehlern* und *Prozeßsystemfehlern*. Als Fehler wird in diesem Zusammenhang das Abweichen eines Systems vom erwarteten spezifizierten Verhalten bezeichnet. Da Rechensystem und Prozeßsystem das betrachtete Gesamtsystem darstellen, ist diese Fehlertypklassifizierung vollständig. Letzteres gilt auch für die Unterscheidung der Ursachen der beiden Fehlertypen, die im folgenden genannt werden.

Rechensystemfehler werden durch Fehlfunktionen der Rechensystemkomponenten verursacht. Gemäß den Annahmen des Abschnitts 2.2.1.4 werden Fehlfunktionen der Hardware und Betriebssystemkernsoftware bereits im Entstehen erkannt. Sie führen zum Abbruch aktiver Berechnungen durch sofortigen Rechensystemstop. Mit diesem verbunden ist (in der Regel) der Verlust des Hauptspeicherinhalts. Da aufgrund der Annahme 2.2.1.4/3 der Sekundärspeicherinhalt im Fehlerfall jedoch in unverfälschter Form erhalten bleibt, hängt im Falle von Rechensystemfehlern die Rekonstruierbarkeit eines konsistenten Wiederaufsetzzustandes nach Auswechslung der fehlerhaften Komponenten bzw. ihrer Reparatur von der realisierten Prozeßsystemzustandssicherungs-Strategie ab. Diesbezügliche Eignung/Wirksamkeit der im Abschnitt 2.2.2 vorgestellten und im weiteren vorausgesetzten Mechanismen bleibt noch zu zeigen.

Im Gegensatz zu Rechensystemfehlern ist die Ursache für *Prozeßsystemfehler* in der Fehlerhaftigkeit von Prozessen hinsichtlich ihres Codes und/oder ihrer Daten zu suchen.
Prozeßsystemfehler werden mittels prozeßsysteminterner Tests (siehe Abschnitt 2.2.4.1) erkannt. Sie äußern sich in einem fehlerhaften Prozeßsystem-Zustand, führen in der Regel jedoch nicht zum Verlust des Hauptspeicherinhalts. Da auch sie nach den Annahmen 2.2.1.4/3 und 4 nicht zu einer Verfälschung oder Löschung des

Sekundarspeicherinhaltes führen können, bleibt auch in diesem Fall zu zeigen, daß die in Abschnitt 2.2.2 skizzierten Datenstrukturen und Zustandssicherungsoperationen zur Wiederherstellung eines konsistenten Wiederaufsetzzustandes nach Fehlerbehebung ausreichen. Die Korrektur eines Prozeßsystemfehlers kann im Falle temporärer/transienter, d.h. nicht zwangslaufig und wiederholbar auftretender Fehler, wie z.B. von Synchronisationsfehlern und Deadlocks, durch Rücksetzung aktueller Berechnungen und Operationswiederholung (in zufällig anderer Reihenfolge) erfolgen. Im Falle von permanenten Fehlern, wie z.B. Programmierfehlern und Variableninitialisierungsfehlern, kann eine Fehlerbehandlung nur durch Ausführung alternativer Operationen beseitigt werden (hierzu vgl. das Recoveryblock-Konzept [HR73], [Hor74], [Ran75], [Ran77]).

Die in diesem Abschnitt verwendeten Begriffe *fehlerhafter* und (*berechnungstechnisch/syntaktisch*) *konsistenter Zustand* lassen sich wie folgt charakterisieren:

Definition 2.2.3.1/1: Ein Zustand eines Prozeßsystems P wird genau dann als (formal) *fehlerhaft* bezeichnet, wenn er vom mittels Tests abgefragten, erwarteten Prozeßsystem-Zustand (in unzulässiger Weise) abweicht oder zu einer durch einen Rechensystemfehler unterbrochenen Berechnung gehört.

Definition 2.2.3.1/2: Ein Zustand eines Prozeßsystems P wird genau dann als (*berechnungstechnisch/syntaktisch*) *konsistent* bezeichnet, wenn er durch eine aktive Berechnung von P erreichbar ist.

Beachte, daß nach Definition 2.2.3.1/2 ein fehlerhafter Prozeßsystem-Zustand durchaus (berechnungstechnisch/syntaktisch) konsistent sein kann. Denn der Begriff der Konsistenz wird in dieser Arbeit nicht mit der Semantik einer Berechnung verknüpft: Jede aktive Aktionenfolge führt zu einem (berechnungstechnisch/syntaktisch) konsistenten Prozeßsystem-Zustand, falls sie nicht durch einen Rechensystemfehler unterbrochen wird. Daß dieser Systemzustand dann evtl. nicht mit dem erwünschten/erwarteten Zustand übereinstimmt und deshalb als (im Vergleich zu einer existierenden Spezifikation) semantisch fehlerhaft bezeichnet wird, liegt nicht an der Ausführung der Berechnungsschritte, sondern an der Implementierung (Codierung und/oder Initialisierung).

Unabhängig von diesem Sachverhalt bleibt für das weitere zusammenfassend festzuhalten, daß von den Auswirkungen her prinzipiell zwei Fehlersituationen, die auch in Kombination auftreten können (nach einem Prozeßsystemfehler tritt, bevor dieser behoben ist, ein Rechensystemfehler auf), zu unterscheiden sind:

(1) *Fehlersituation 1:* Eine Berechnung wird durch einen Rechensystemfehler unterbrochen. Der Hauptspeicherinhalt geht verloren.

(2) *Fehlersituation 2:* Während einer Berechnung führen prozeßsysteminterne Tests zur Erkennung eines Prozeßsystemfehlers, d.h. der Fehlerhaftigkeit eines Prozeßsystem-Zustandes, wobei globale Variablen und/oder lokale Prozeßzustände betroffen sein können.

Da z.B. ein Fehlerbehandlungsvorgang vor seinem erfolgreichen Abschluß durch einen Rechensystemfehler unterbrochen werden kann, ist prinzipiell auch zu berücksichtigen, daß mehrere Fehlersituationen gleichzeitig auftreten können. Die Fehlerbehebung kann dann mittels der im nächsten Abschnitt beschriebenen Wiederanlaufprozedur sowohl in mehreren Einzelschritten als auch in einem Schritt erfolgen. Dabei ist die Reihenfolge, in der die Fehlerbeseitigung durchgeführt wird, belanglos.

2.2.3.2. Allgemeine Wiederanlaufprozedur

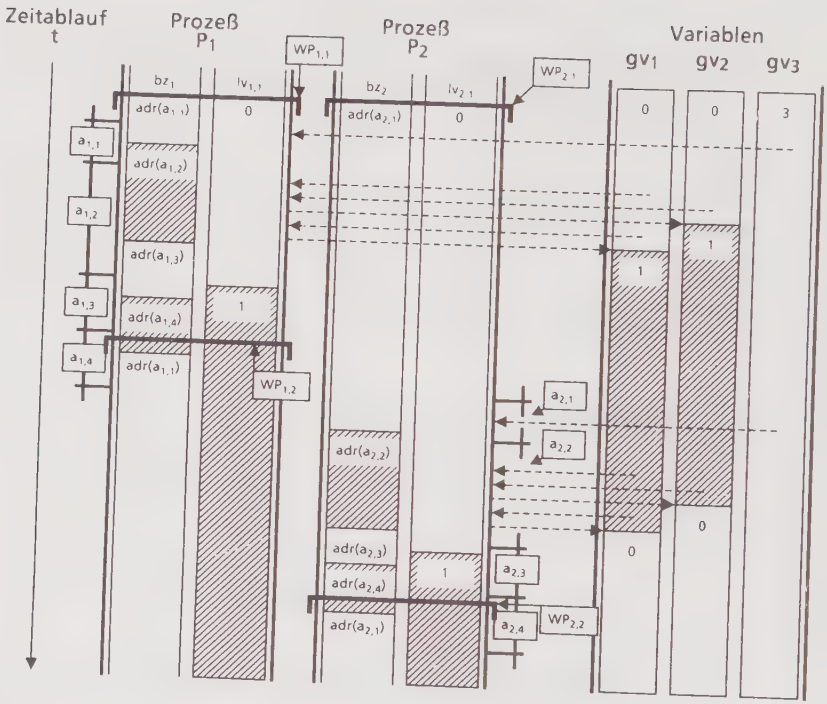
Eine Wiederanlaufprozedur muß unter Berücksichtigung der Forderung nach Wiederherstellung eines konsistenten Prozeßsystem-Zustandes folgende Fehlerbehandlungs-Strategien realisieren:

(1) *Rücksetzung aller Prozesse* P_i eines Prozeßsystems zu dem jeweils aktuellsten Wiederaufsetzpunkt, d.h. zu dem in $h(P_i)[|h(P_i)|-1]$ gesicherten, prozeßlokalen Zustand,

(2) gezielte *Rücksetzung eines Prozesses (mehrerer Prozesse)* zu einem (jeweils) bestimmten Wiederaufsetzpunkt, wie etwa des Prozesses P_i zum Wiederaufsetzpunkt $WP_{i,j}$, der dem Feldelement $h(P_i)[j]$ der Wiederaufsetzpunkthistorie zu Prozeß P_i entspricht, und

(3) gezielte *Wiederherstellung eines/mehrerer globalen Variablenwerte(s)*, z.B. Wiederherstellung des in $h(gv_k)[l]$ gesicherten Variablenwertes von gv_k.

Bei Zugrundelegung des in Bild 2.2.3.2/1 schematisch dargestellten Berechnungsablaufs des Beispiel-Prozeßsystems P (vgl. Bild 2.2.2.4/1) lassen sich die Forderungen nach Unterstützung der Rücksetzung aller Prozesse eines Prozeßsystems zu dem jeweils aktuellsten Wiederaufsetzpunkt, nach gezielter Rücksetzung eines Prozesses (mehrerer Prozesse) zu einem (jeweils) bestimmten Wiederaufsetzpunkt und nach gezielter Wiederherstellung eines/mehrerer globalen Variablenwerte(s) anhand der Bilder 2.2.3.2/2, 3 bzw. 4 visualisieren.



Legende:

$a_{i,j}$ .. = j'te Aktion des Prozesses P_i muß zurückgesetzt werden, da gelesene Variablenwerte durch Rücksetzung anderer Aktionen entzogen werden;

$WP_{i,j}$ :: = j'ter Wiederaufsetzpunkt des Prozesses P_i;

bz_i : = Befehlszähler des Prozesses P_i;

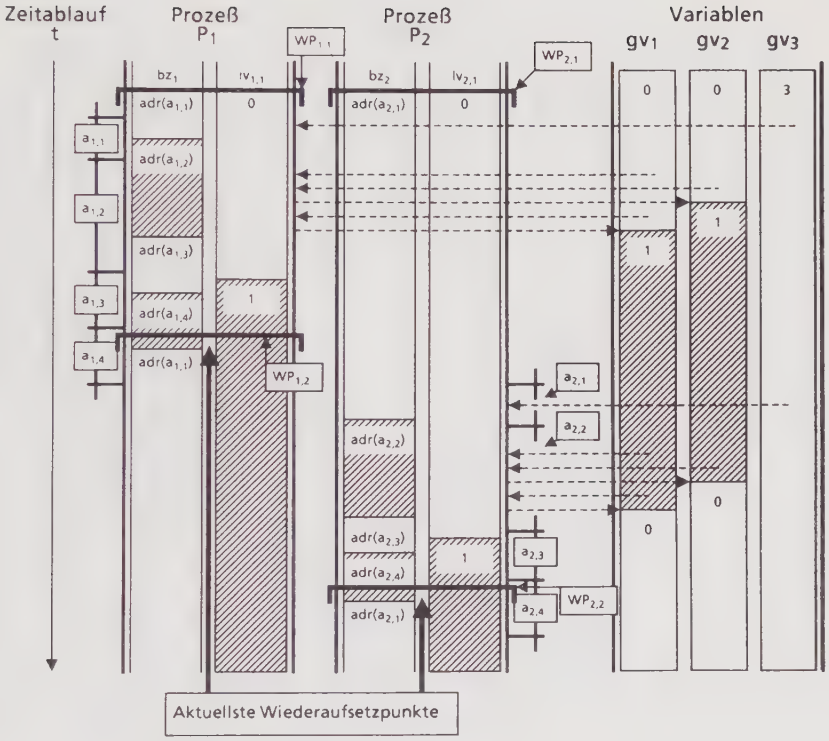
$lv_{i,j}$ :: = j'te lokale Variable des Prozesses P_i;

gv_j :: = j'te globale Variable;

$adr(a_{i,j})$ :: = Adresse der j'ten Aktion des Prozesses P_i;

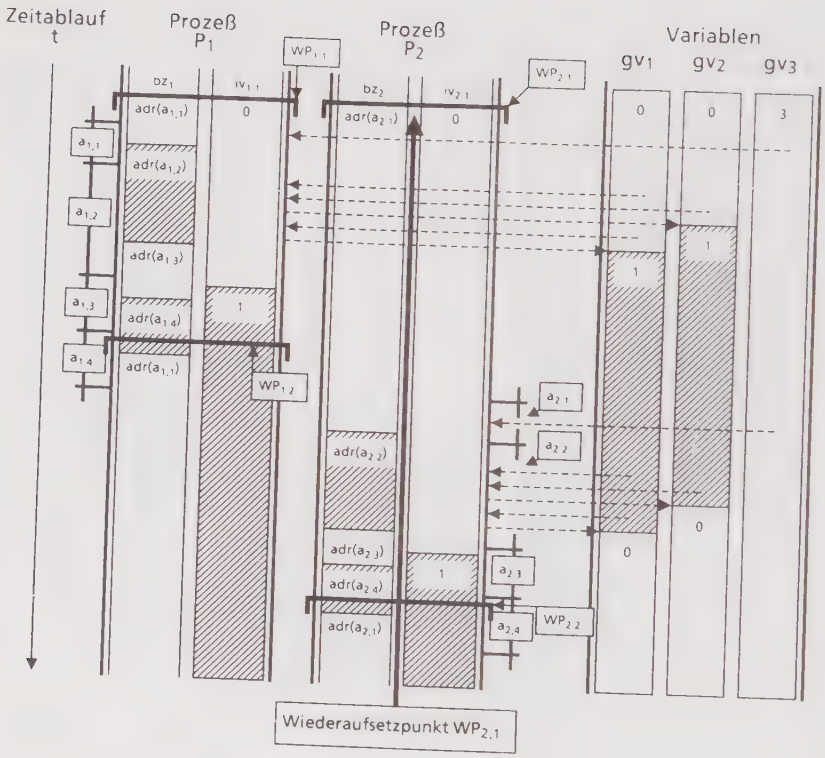
Übergänge von ▭ nach ▨ und von ▨ nach ▭ verdeutlichen Variablenwertänderungen;

Bild 2.2.3.2/1: Wiederaufsetzpunktgesicherter Berechnungsablauf
$a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4}$



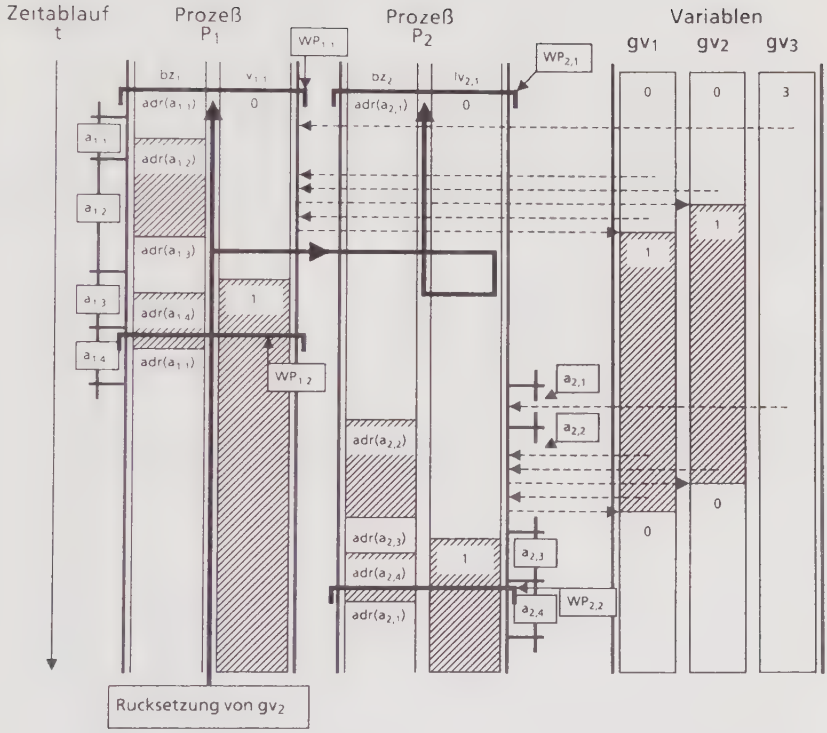
Legende: siehe Bild 2.2.3.2/1;

Bild 2.2.3.2/2: Rücksetzung aller Prozesse zum aktuellsten Wiederaufsetzpunkt



Legende: siehe Bild 2.2.3.2/1.

Bild 2.2.3.2/3: Rücksetzung von Prozeß P_2 zum Wiederaufsetzpunkt $WP_{2,1}$



Legende: siehe Bild 2 2 3 2/1;

Bild 2.2.3.2/4: Rücksetzung des in Prozeß P_1 der Variablen gv_2 zugewiesenen Wertes

Eine Wiederanlaufprozedur, welche die drei obengenannten Forderungen erfüllt und auf dem in Abschnitt 2.2.2 vorgestellten Prozeßsystem-Zustandssicherungskonzept aufbaut, stellt Algorithmus 2.2.3.2/1 dar.
Nach Eingabe derjenigen Wiederaufsetzpunkte, zu denen Prozesse und globale Variablen eines Prozeßsystems zurückgesetzt werden sollen, in Schritt 1, errechnet die Wiederanlaufprozedur im 2. Schritt welche zusätzlichen Rücksetzschritte erforderlich sind, um einen im Sinne von Definition 2.2.3.1/2 konsistenten Prozeßsystem-Zustand incl. der entsprechenden Zustands-Historie wiederherzustellen. Die Rekonstruktion eines konsistenten Gesamtzustandes erfolgt anschließend in den Schritten 3 und 4. Im dritten Schritt bezogen auf die Zustands-Historie und in Schritt vier hinsichtlich des zugehörigen aktuellen Prozeßsystem-Zustandes.

Die Ermittlung eines konsistenten Zustandes nach Rücksetzung aller Prozesse eines Prozeßsystems P kann mit Hilfe von Algorithmus 2.2.3.2/1 durch Initialisierung des 0'ten Elementes $LWPR[0]$ des Feldes lokaler Wiederaufsetzpunktreferenzen $LWPR[\,]$ mit der jeweiligen Adresse des aktuellsten Wiederaufsetzpunktes $h(P_i)[|h(P_i)|-1]$ eines jeden Prozesses P_i von P aktiviert werden.
Gezielte Rücksetzung eines Prozesses (mehrerer Prozesse) P_i zu einem (jeweils) bestimmten Wiederaufsetzpunkt $WP_{i,j}$ kann mittels Algorithmus 2.2.3.2/1 durch Zuweisung des Zeigers $\uparrow h(P_i)[j]$ zur Menge $LWPR[0]$ eingeleitet werden, und gezielte Wiederherstellung eines/mehrerer globalen Variablenwerte(s), wie z.B. des Wertes $h(gv_k)[l].gv_k$, kann durch Erweiterung der Menge $LWPR[0]$ um das Element $h(gv_k)[l].gSR$ initiiert werden.

Von der als Algorithmus 2.2.3.2/1 bezeichneten Wiederanlaufprozedur werden potentielle Fehlerverursacher nicht berücksichtigt, d.h. die Wiederanlaufprozedur ermittelt nicht alle für eine auszuführende Prozeßsystemzustands-Rücksetzung potentiell verantwortlich zu machenden fehlerhaften Variablenwerte, sondern versucht lediglich, unter Berücksichtigung der Rücksetzanforderung, den mit dem geringsten Verlust an Berechnungsschritten verbundenen konsistenten Zustand zu rekonstruieren. Eine Berücksichtigung potentieller Fehlerverursacher wird erst mit der Einführung einer Verifikationsanzeige für jeden globalen Variablenwert und je Wiederaufsetzpunkt möglich (siehe Abschnitte 2.2.4.1 und 2.2.4.2).

Algorithmus 2.2.3.2/1 (Wiederanlaufprozedur bei Nichtberücksichtigung potentieller Fehlerverursacher):

```
/* Algorithmus-BEGINN */

Schritt 0:  /* Variablen-Deklaration */

integer l, l1, l2;

for all Pj from P:      {struct element_of h(Pj) h(Pj)[];};
for all gvk ∈ gVP:      {struct element_of h(gv) h(gvk)[];};

set_of struct ↑element_of h(Pi) LWPR[];
set_of struct ↑element_of h(gv) LgSR[], LgLR;

Schritt 1:  /* Initialisierung; für alle Pj aus P und gvk∈gVP wird angenommen, daß
               h(Pj) bzw. h(gvk) die Wiederaufsetzpunkthistorie zu Pj bzw. gvk
               repräsentieren und bereits entsprechend initialisiert sind */

l := 0;

LWPR[0] := { ↑h(Pj)[k] l   ∃ h(Pj)[m]: Pj muß zum Wiederaufsetzpunkt h(Pj)[m]
                           zurückgesetzt werden & m ≤ k
                        ∥
                           ∃ h(gvm)[n]: der in h(gvm)[n] gesicherte Wert wurde als
                           fehlerhaft erkannt & h(gvm)[n].gSR = ↑h(Pj)[k]
           };

Schritt 2:  /* Ermittlung der Referenzen aller bei der Wiederherstellung eines
               konsistenten Berechnungszustandes potentiell zu berücksichtigenden
               Elemente der Zustands-Historie zu P */

do {

Schritt 2.1:    LgSR[l]   = { ↑h(gvm)[n] l  ∃ ↑h(Pj)[k]∈LWPR[l]: ↑h(gvm)[n]∈h(Pj)[k].LgSR};
                l := l + 1;

Schritt 2.2:    LWPR[l]   := { ↑h(Pj)[k]  l  ∃ ↑h(gvm)[n]∈LgSR[l-1]: ↑h(Pj)[k]∈h(gvm)[n].LgLR};

Schritt 2.3:    LWPR[l]   := LWPR[l-1] ∪ { ↑h(Pj)[k] l  ∃ ↑h(Pj)[m]∈LWPR[l] & m ≤ k};

} while LWPR[l-1] ⊂ LWPR[l];

Schritt 3:  /* Herstellung einer zum Wiederaufsetzzustand zu P konsistenten
               Zustands-Historie */

Schritt 3.1:  LgLR := { ↑h(gvm)[n] l  ∃ ↑h(Pj)[k]∈LWPR[l]: ↑h(gvm)[n]∈h(Pj)[k].LgLR};

Schritt 3.2:  for all ↑h(gvk)[m]∈LgLR:  {h(gvk)[m].LgLR: = h(gvk)[m].LgLR - LWPR[l];};

Schritt 3.3:  for all gvk∈gVP:  {
                    l2 := 0;
                    from l1 := 0 step 1 until l1 = lh(gvk)l-1
                        { if ↑h(gvk)[l1]∉LgSR[l-1] then  {
                                                          h(gvk)[l2]: = h(gvk)[l1];
                                                          l2: = l2 + 1;
                                                        };
                        };
                    h(gvk): = h(gvk)[0,...,l2-1];
                };
```

```
Schritt 3.4: for all Pj from P: {
                 if ∃k: ↑h(Pj)[k]∈LWPR[l] then
                 {
                     h(Pj): = h(Pj)[0,...,min{k| h(Pj)[k]∈LWPR[l]}];
                     h(Pj)[min{k| h(Pj)[k]∈LWPR[l]}].LgSR: = "*";
                     h(Pj)[min{k| h(Pj)[k]∈LWPR[l]}].LgLR: = "*";
                 };
             };

Schritt 4:   /* Herstellung eines konsistenten Wiederaufsetzzustandes durch
             Initialisierung des Datensegmentes von P nach folgender Vorschrift */

Schritt 4.1: for all Pj from P: {
                 if ∃k: ↑h(Pj)[k]∈LWPR[l] then
                 {
                     for all lv∈VPj: {
                                         lv: = last(h(Pj)).lv;
                                     };
                 };
             };

Schritt 4.2: for all gvk∈gVP: { gvk: = last(h(gvk)).gvk };

             /* Algorithmus-ENDE */
```

2.2.3.3. Erläuterung am Beispiel

Die beiden Beispiele 2.2.3.3/1 und 2 veranschaulichen exemplarisch die Einzelschritte, welche zur Wiederherstellung eines konsistenten Zustandes bei Rücksetzung des Prozesses P_1 zum Wiederaufsetzpunkt $h(P_1)[0]$ (bzw. bei Rücksetzung des in $h(gv_2)[2]$ gesicherten Variablenwertes) im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ des Beispiel-Prozeßsystems P (vgl. Bild 2.2.2.4/10) und Verwendung des Algorithmus 2.2.3.2/1 durchlaufen werden müssen.
In den Bildern 2.2.3.3/1 und 2 sind die in den Beispielen 2.2.3.3/1 bzw. 2 errechneten konsistenten Zustände wiedergegeben.

Beispiel 2.2.3.3/1 (Zurücksetzung von P_1 nach $h(P_1)[0]$ im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$):

Schritt 1:
```
LWPR[0]    :=    {↑h(P1)[0], ↑h(P1)[1]};
```

Schritt 2:
```
/*********************************************************************/
```

Schritt 2.1:
```
LgSR[0]    :=    {↑h(gv1)[1], ↑h(gv2)[1]};
```

Schritt 2.2:
```
LWPR[1]    :=    {↑h(P2)[0]};
```

Schritt 2.3:
```
LWPR[1]    :=    {↑h(P1)[0], ↑h(P1)[1], ↑h(P2)[0], ↑h(P2)[1]};
```

Schritt 2.1:
```
LgSR[1]    :=    {↑h(gv1)[1], ↑h(gv2)[1], ↑h(gv1)[2], ↑h(gv2)[2]};
```

Schritt 2.2:
```
LWPR[2]    :=    {↑h(P2)[0]};
```

Schritt 2.3:
```
LWPR[2]    :=    {↑h(P1)[0], ↑h(P1)[1], ↑h(P2)[0], ↑h(P2)[1]} =    LWPR[1];
```

Schritt 3:
```
/*********************************************************************/
```

Schritt 3.1:
```
LgLR       :=    {↑h(gv1)[0], ↑h(gv2)[0], ↑h(gv3)[0], ↑h(gv1)[1], ↑h(gv2)[1]};
```

Schritt 3.2:
```
h(gv1)[0].LgLR := "*";          h(gv1)[1].LgLR := "*";
h(gv2)[0].LgLR := "*";          h(gv2)[1].LgLR := "*";
h(gv3)[0].LgLR := "*";
```

Schritt 3.3:
```
h(gv1) := h(gv1)[0];
h(gv2) := h(gv2)[0];
h(gv3) := h(gv3)[0];
```

Schritt 3.4:
```
h(P1) := h(P1)[0];      h(P1)[0].LgSR := "*";      h(P1)[0].LgLR := "*";
h(P2) := h(P2)[0];      h(P2)[0].LgSR := "*";      h(P2)[0].LgLR := "*";
```

Schritt 4:
```
/*********************************************************************/
```

Schritt 4.1:
```
bz1 := adr(a1,1);    lv1,1 := 0;        bz2 := adr(a2,1);    lv2,1 := 0;
```

Schritt 4.2:
```
gv1 := 0;        gv2 := 0;        gv3 := 3;
```

Beispiel 2.2.3.3/2 (Im Zustand $val_p(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ erforderliche Rücksetzausbreitung bei Erkennung des in $h(gv_2)[2]$ gesicherten Wertes von gv_2 als fehlerhaft):

Schritt 1:
```
LWPR[0]    := { ↑h(P2)[0], ↑h(P2)[1] };
```

Schritt 2:
```
/*************************************************************************/
```

Schritt 2.1:
```
LgSR[0]    := { ↑h(gv1)[2], ↑h(gv2)[2]};
```

Schritt 2.2:
```
LWPR[1]    := ∅;
```

Schritt 2.3:
```
LWPR[1]    := { ↑h(P2)[0], ↑h(P2)[1]} = LWPR[0];
```

Schritt 3:
```
/*************************************************************************/
```

Schritt 3.1:
```
LgLR       := { ↑h(gv3)[0], ↑h(gv1)[1], ↑h(gv2)[1]};
```

Schritt 3.2:
```
h(gv1)[1].LgLR := "*";
h(gv2)[1].LgLR := "*";
h(gv3)[0].LgLR := { ↑h(P1)[0]};
```

Schritt 3.3:
```
h(gv1) := h(gv1)[0,1];
h(gv2) := h(gv2)[0,1];
h(gv3) := h(gv3)[0];
```

Schritt 3.4:
```
h(P1) := h(P1)[0,1];
h(P2) := h(P2)[0];          h(P2)[0].LgSR := "*";      h(P2)[0].LgLR := "*";
```

Schritt 4:
```
/*************************************************************************/
```

Schritt 4.1:
```
bz2 := adr(a2,1);     lv2,1 := 0;
```

Schritt 4.2:
```
gv1 := 1;             gv2 := 1;          gv3 := 3;
```

Bild 2.2.3.3/1: Im Beispiel 2.2.3.3/1 ermittelter Wiederaufsetzzustand (Beispiel-Prozeßsystem-Zustand $val_P(\lambda)$)

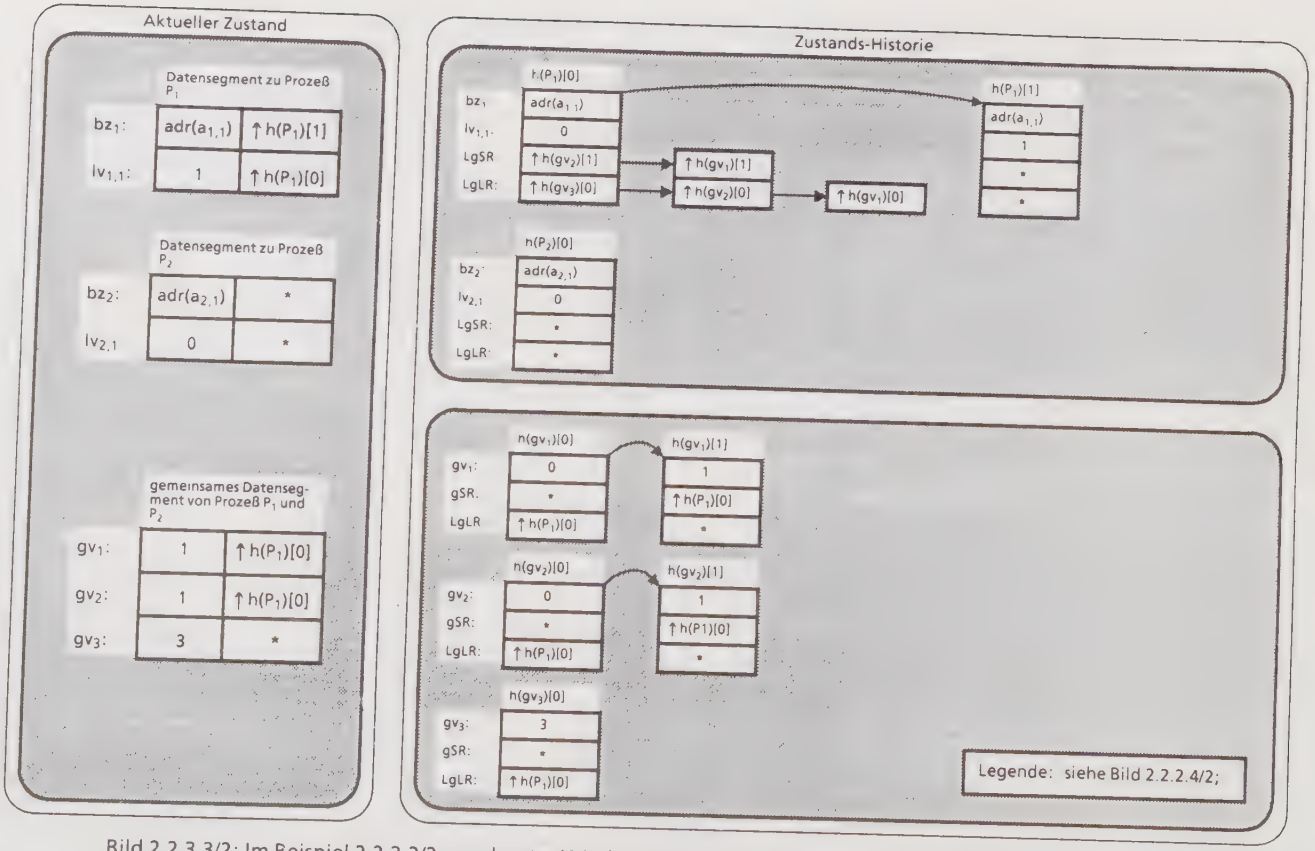


Bild 2.2.3.3/2: Im Beispiel 2.2.3.3/2 errechneter Wiederaufsetzzustand (Beispiel-Prozeßsystem-Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4})$)

2.2.3.4. Petri-Netz-Darstellung eines Fehlerbehandlungsvorgangs

Anschaulicher als im vorangehenden Abschnitt läßt sich der Rücksetzvorgang anhand eines Petri-Netzes darstellen. Dazu ist die Abbildung der Zustands-Historie auf einen Petri-Netz-Graphen erforderlich. Eine Darstellung der Zustands-Historie in Form eines Petri-Netzes bietet den zusätzlichen Vorteil neu einzuführende Begriffe, auf die später zurückgegriffen wird, exakt erläutern und motivieren zu können.

Bemerkung: Der mit Petri-Netzen vertraute Leser kann die folgenden Definitionen grundlegender Begriffe aus der Petri-Netz-Theorie, inclusive ihrer Erläuterung anhand von Beispiel 2.2.3.4/1, überspringen.

Definition 2.2.3.4/1: Ein *Petri-Netz* P ist ein Tripel (S,T,K), bestehend aus zwei disjunkten endlichen Mengen S (von Stellen), T (von Transitionen) und einer Menge K (von gerichteten Kanten), $K \subseteq (S \times T) \cup (T \times S)$.

Definition 2.2.3.4/2: Sei $P=(S,T,K)$ ein Petri-Netz:

(1) Eine Stelle $s \in S$ heißt *Eingabestelle* einer Transition $t \in T$, falls $(s,t) \in K$ ist.

(2) Eine Stelle $s \in S$ heißt *Ausgabestelle* einer Transition $t \in T$, falls $(t,s) \in K$ ist.

(3) Eine Transition $t \in T$ heißt *Eingabetransition* einer Stelle $s \in S$, falls $(t,s) \in K$ ist.

(4) Eine Transition $t \in T$ heißt *Ausgabetransition* einer Stelle $s \in S$, falls $(s,t) \in K$ ist.

(5) Die *Vorgängermenge einer Stelle* $s \in S$ wird mit ${}^\bullet s$ bezeichnet:

$${}^\bullet s ::= \{t \in T \mid (t,s) \in K\}.$$

(6) Die *Nachfolgermenge einer Stelle* $s \in S$ wird mit $s^\bullet$ bezeichnet:

$$s^\bullet ::= \{t \in T \mid (s,t) \in K\}.$$

(7) Die *Vorgängermenge einer Transition* $t \in T$ wird mit ${}^\bullet t$ bezeichnet:

$${}^\bullet t ::= \{s \in S \mid (s,t) \in K\}.$$

(8) Die *Nachfolgermenge einer Transition* $t \in T$ wird mit $t^\bullet$ bezeichnet:

$$t^\bullet ::= \{s \in S \mid (t,s) \in K\}.$$

(9) Eine Stelle $s \in S$ heißt *Konfliktstelle*, falls $|s^\bullet| > 1$ ist.

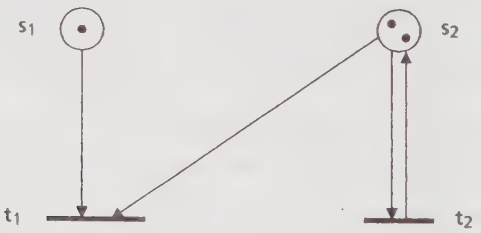
Definition 2.2.3.4/3: Eine *Markierung* eines Petri-Netzes $P=(S,T,K)$ ist eine Funktion $M: S \to \mathbf{N}_0$. P in Verbindung mit M heißt *markiertes Petri-Netz* (P,M).

Definition 2.2.3.4/4: Sei ein markiertes Petri-Netz (P,M) gegeben. Eine Transition $t \in T$ *kann schalten* genau dann, wenn $M(s)>0$ für alle $s \in {}^\bullet t$ gilt. Das Schalten von t führt die Markierung M in eine Markierung M' über, wobei gilt:

$$M'(s) ::= \begin{cases} M(s)-1 & s \in ({}^\bullet t - t^\bullet) \\ M(s)+1 & falls \quad s \in (t^\bullet - {}^\bullet t) \\ M(s) & sonst \end{cases}$$

Petri-Netze werden in der Literatur meistens graphisch dargestellt. Dabei werden stets Stellen durch Kreise, Transitionen durch Balken und Kanten durch sie verbindende Pfeile repräsentiert.

Beispiel 2.2.3.4/1: Der folgende Graph stellt ein markiertes Petri-Netz dar:



s_1, s_2 sind Eingabestellen von t_1, s_2 ist Eingabe- und Ausgabestelle von t_2.
${}^\bullet s_1 = \phi$, $s_1^\bullet = \{t_1\}$, ${}^\bullet s_2 = \{t_2\}$, $s_2^\bullet = \{t_1, t_2\}$.
Das Petri-Netz ist markiert. Dies wird durch die Punkte "•" in den Stellen s_1 und s_2 kenntlich gemacht. $M(s_1)=1$, $M(s_2)=2$.
Beide Transitionen können schalten. Beim Schalten der Transition t_1 wird von den beiden Eingabestellen dieser Transition jeweils 1 Marke abgezogen. Beim Schalten von t_2 würde sich die Markierung nicht ändern, da die einzige Eingabestelle der Transition t_2 auch ihre einzige Ausgabestelle ist.
Die Stelle s_2 ist eine Konfliktstelle.

Man überlegt sich leicht, daß man eine Transition in einem Petri-Netz auch als synchronisierende Aktion darstellen kann. Beispielsweise kann die Transition t_1 aus Beispiel 2.2.3.4/1 interpretiert werden als eine Aktion t_1:

$$t_1 ::= \textit{when}\ \ bz_i = adr(t_1)\ \&\ M(s_1)>0\ \&\ M(s_2)>0$$
$$\textit{do}\ \ bz_i := adr(t_1)';\ M'(s_1) := M(s_1)-1;\ M'(s_2) := M(s_2)-1;\ \textit{od};$$

Allgemein kann eine Transition t repräsentiert werden durch:

$$t ::= \textit{when}\ \ bz_j = adr(t)\ \&\ \bigcap_{i\in A} s_i > 0$$
$$\textit{do}\ \ bz_j := adr(t)';\ \underset{i\in B}{s_i := s_i - 1};\ \underset{i\in C}{s_i := s_i + 1};\ \textit{od};$$

wobei $s_i = M(s_i)$ und A, B, C Teilmengen der Menge $\{1,\ldots,m\}$ sind, wenn das zugehörige Petri-Netz m Stellen $s_1, \ldots, s_m$ hat. A ist die Menge der Indizes der Eingabestellen von t. B ist die Menge der Indizes der Eingabestellen von t, die nicht gleichzeitig auch Ausgabestellen von t sind, und C ist die Menge der Indizes der Ausgabestellen von t, die nicht gleichzeitig Eingabestellen von t sind. Daher gilt $B \subseteq A$ und $A \cap C = \phi$.

Durch Abstraktion kann zu jeder Zustands-Historie in folgender eindeutiger Weise ein Petri-Netz konstruiert werden:

(1) Jedes Element $h(P_i)[j]$ wird durch drei Stellen s_1, s_2, s_3 und drei Transitionen t_1, t_2, t_3 repräsentiert, eine exakte Definition von t_1, t_2 und t_3 gibt Tabelle 2.2.3.4/1 wieder, in der statt der ausführlichen Schreibweise $M(s_i)$, wie auch im weiteren, abkürzend die Notation s_i verwendet wird, und

(2) jedem Element $h(gv_k)[l]$ werden drei Stellen s_1, s_2, s_3 und zwei Transitionen t_1 und t_2 zugeordnet, eine Beschreibung von t_1 und t_2 enthält Tabelle 2.2.3.4/2.

Mit $h(P_i)[j].s_k$ und $h(P_i)[j].t_k$ wird im folgenden auf die Stelle s_k bzw. die Transition t_k der Petri-Netz-Repräsentation von $h(P_i)[j]$ bezug genommen, $k \in \{1,2,3\}$. Analoges gilt für die Bezeichnung der Stellen und Transitionen, welche in Verbindung zu $h(gv_k)[l]$ gebracht werden.

Ein nach der eben skizzierten Vorschrift zu der in Bild 2.2.2.4/10 dargestellten Zustands-Historie konstruiertes Petri-Netz zeigt Bild 2.2.3.4/1. Um die Darstellung übersichtlich zu gestalten, wurden die sich nach den Definitionen der Tabellen 2.2.3.4/1 und 2 teilweise überlappenden Transitionen der Elemente der Historie $h(P_1)$, $h(P_2)$, $h(gv_1)$, $h(gv_2)$ und $h(gv_3)$ "auseinandergezogen" dargestellt. Die Legende zu Bild 2.2.3.4/1 zeigt, wie die stellenlosen Verbindungen zwischen zwei Transitionen zu interpretieren sind, um zu einem der Definition 2.2.3.4/1 genügenden Petri-Netz zu kommen, welches ein, zu einem gemäß der Definitionen der Tabellen 2.2.3.4/1 und 2

Aktionsname	Φ	Θ
$h(P_i)[j].t_1$ für $j>0$	$\Phi(h(P_i)[j-1].t_3)$	$h(P_i)[j].s_1 := h(Pi)[j].s_1 + 1;$
$h(P_i)[j].t_2$	$\bigcup_{\uparrow h(gv_k)[l] \in h(P_i)[j].LgLR} \Phi(h(gv_k)[l].t_2)$ ‖ $\bigcup_{\uparrow h(gv_k)[l] \in h(P_i)[j].LgSR} \Phi(h(gv_k)[l].t_2)$ ‖ $\Phi(h(P_i)[j-1].t_3)$	$h(P_i)[j].s_2 := h(Pi)[j].s_2 + 1;$
$h(P_i)[j].t_3$	$h(P_i)[j].s_2>0$ & $h(P_i)[j].s_3>0$	$\forall \uparrow h(gv_k)[l] \in h(P_i)[j].LgSR: \Theta(h(gv_k)[l].t_1);$ $\Theta(h(P_i)[j+1].t_1);$ $\Theta(h(P_i)[j+1].t_2);$ $h(P_i)[j].s_2 := h(Pi)[j].s_2 - 1;$ $h(P_i)[j].s_3 := h(Pi)[j].s_3 - 1;$

Tabelle 2.2.3.4/1: $h(P_i)[j]$ zugeordnete Transitionen t_1, t_2 und t_3

Aktionsname	Φ	Θ
$h(gv_k)[l].t_1$ für $l>0$	$h(gv_k)[l].gSR = \uparrow h(P_i)[j]: \Phi(h(P_i)[j].t_3)$	$h(gv_k)[l].s_1 := h(gv_k)[l].s_1 + 1;$ $h(gv_k)[l].s_2 := h(gv_k)[l].s_2 + 1;$
$h(gv_k)[l].t_2$	$h(gv_k)[l].s_2>0$ & $h(gv_k)[l].s_3>0$	$\forall \uparrow h(P_i)[j] \in h(gv_k)[l].LgLR: \Theta(h(P_i)[j].t_2);$ $\uparrow h(P_i)[j] \in h(gv_k)[l].gSR: \Theta(h(P_i)[j].t_2);$ $h(gv_k)[l].s_2 := h(gv_k)[l].s_2 - 1;$ $h(gv_k)[l].s_3 := h(gv_k)[l].s_3 - 1;$

Tabelle 2.2.3.4/2: $h(gv_k)[l]$ repräsentierende Transitionen t_1 und t_2

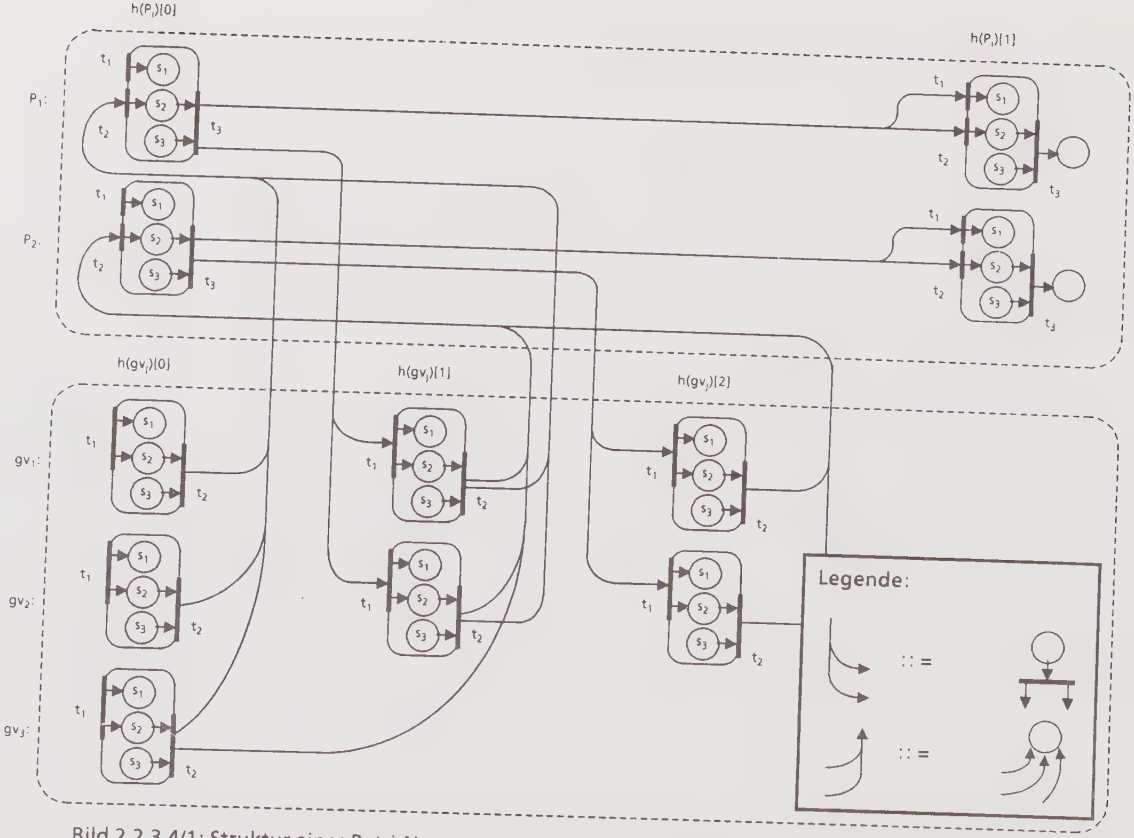


Bild 2.2.3.4/1: Struktur eines Petri-Netzes zur Darstellung der Zustands-Historie von Bild 2.2.2.4/10

konstruierten Petri-Netz, äquivalentes Verhalten zeigt.

Bemerkung: Da Prozesse nicht vor ihren Anfangszustand zurückgesetzt werden können, ist ergänzend zu den Angaben in den Tabellen 2.2.3.4/1 und 2, in den folgenden Petri-Netz-Repräsentationen $\Phi(h(P_i)[0].t_1)$ als **false** und $\Phi(h(gv_k)[0].t_1)$ ebenso als **false** zu interpretieren.

Um einen Rücksetzvorgang mittels des zur Zustands-Historie des Bildes 2.2.2.4/10 strukturäquivalenten Petri-Netz-Graphen, vgl. Bild 2.2.3.4/1, veranschaulichen zu können, muß das Petri-Netz zunächst, wie in Bild 2.2.3.4/2 wiedergegeben, initialisiert werden. Dabei werden nur die Stellen s_3 markiert. Diese Markierung stellt sicher, daß alle Transitionen des Typs $h(P_i)[j].t_3$ und des Typs $h(gv_k)[l].t_2$ nur maximal einmal schalten können. Dadurch ist es möglich, als Endekriterium eines mittels eines Petri-Netzes dargestellten Rücksetzvorgangs einen Zustand zu wählen, in dem das Petri-Netz nicht mehr lebendig ist. Ein Petri-Netz-Zustand wird als *lebendig* bezeichnet, wenn in ihm mindestens eine Transition schalten kann (ein Petri-Netz ist in einem Zustand $val_P(\alpha)$ lebendig genau dann, wenn $lf_P(\alpha) \neq \phi$).
Der Wunsch nach Rücksetzung eines Historienelementes kann durch Markierung der Stelle s_2 dieses Historienelementes zum Ausdruck gebracht werden.
Ein Übergang von einem Petri-Netz-Zustand $val_P(\alpha)$ in einen Folgezustand $val_P(\alpha)'$ wird genau dann als *ein Rücksetzschritt* bezeichnet, wenn alle in $val_P(\alpha)$ lauffähigen Transitionen in einem Takt durchschalten.
Wird während eines Rücksetzvorgangs ein Zustand $val_P(\alpha)$ erreicht, für den $lf_P(\alpha)=\phi$ gilt, d.h. der nicht lebendig ist, so zeigen die markierten Stellen s_1 an, welche Elemente der Historien gelöscht werden müssen, um einen konsistenten Berechnungszustand zu erhalten. Die Historienelemente mit dem jeweils höchsten Index pro Historie mit nicht markierten Stellen s_1 repräsentieren den aktuellsten konsistenten Berechnungszustand.

Die Bilderfolge 2.2.3.4/3 bis 6 zeigt, daß im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ bei Zugrundelegung der Zustands-Historie von Bild 2.2.2.4/10 und Erkennung von $h(gv_2)[2]$ als fehlerhaft, dies wird durch Markierung der Stelle $h(gv_2)[2].s_2$ zum Ausdruck gebracht (siehe Bild 2.2.3.4/3), vier Rücksetzschritte zur Errechnung eines konsistenten Zustandes erforderlich sind. Das in Bild 2.2.3.4/6 skizzierte Endresultat des Rücksetzvorganges (die Historien-Elemente $h(P_2)[1]$, $h(gv_1)[2]$ und $h(gv_2)[2]$ müssen gelöscht werden) stimmt mit dem in Beispiel 2.2.3.3/2 abgeleiteten Ergebnis überein.

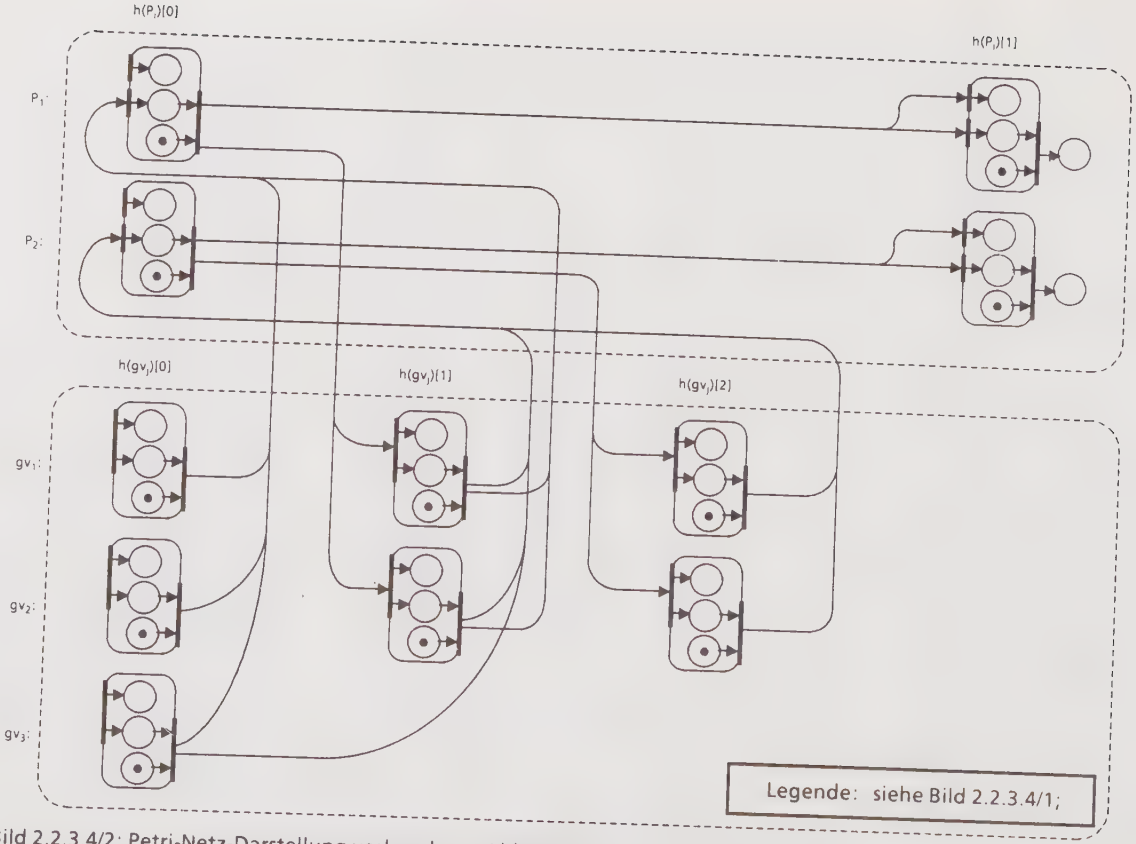


Bild 2.2.3.4/2: Petri-Netz-Darstellung vorhandener Abhängigkeiten im Zustand *valp*($a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4}$)

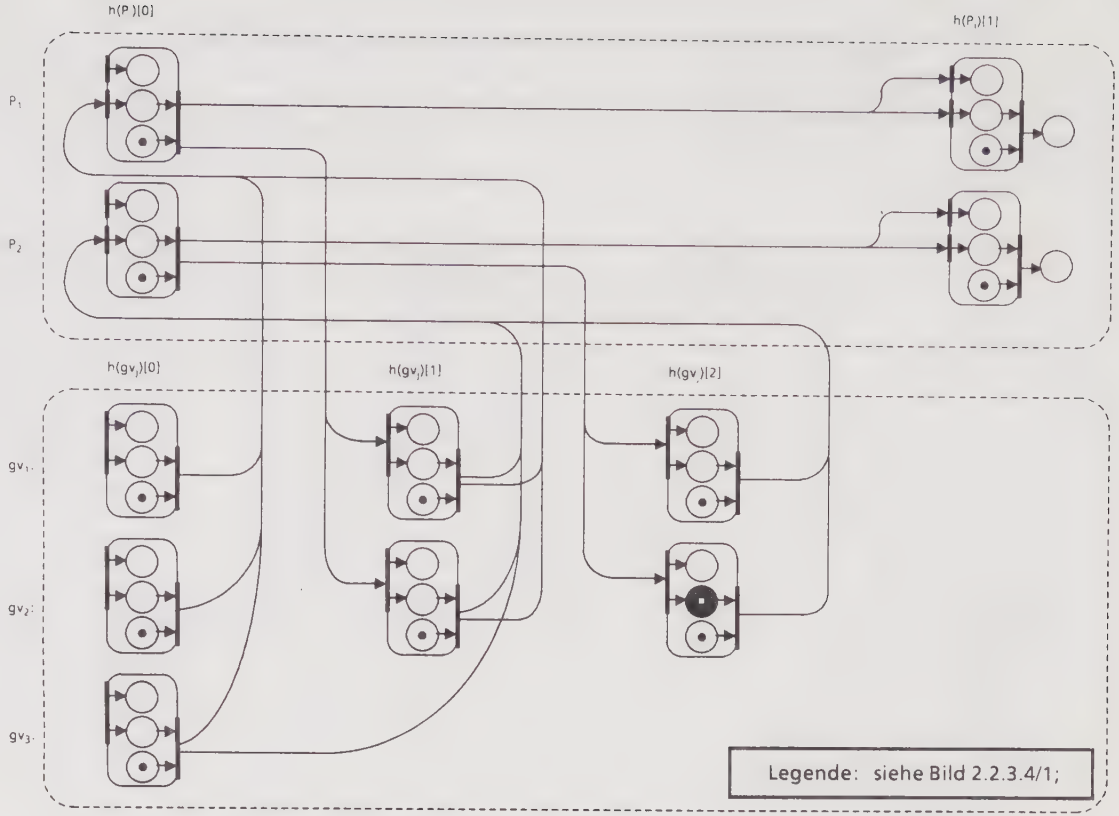


Bild 2.2.3.4/3: 1. Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

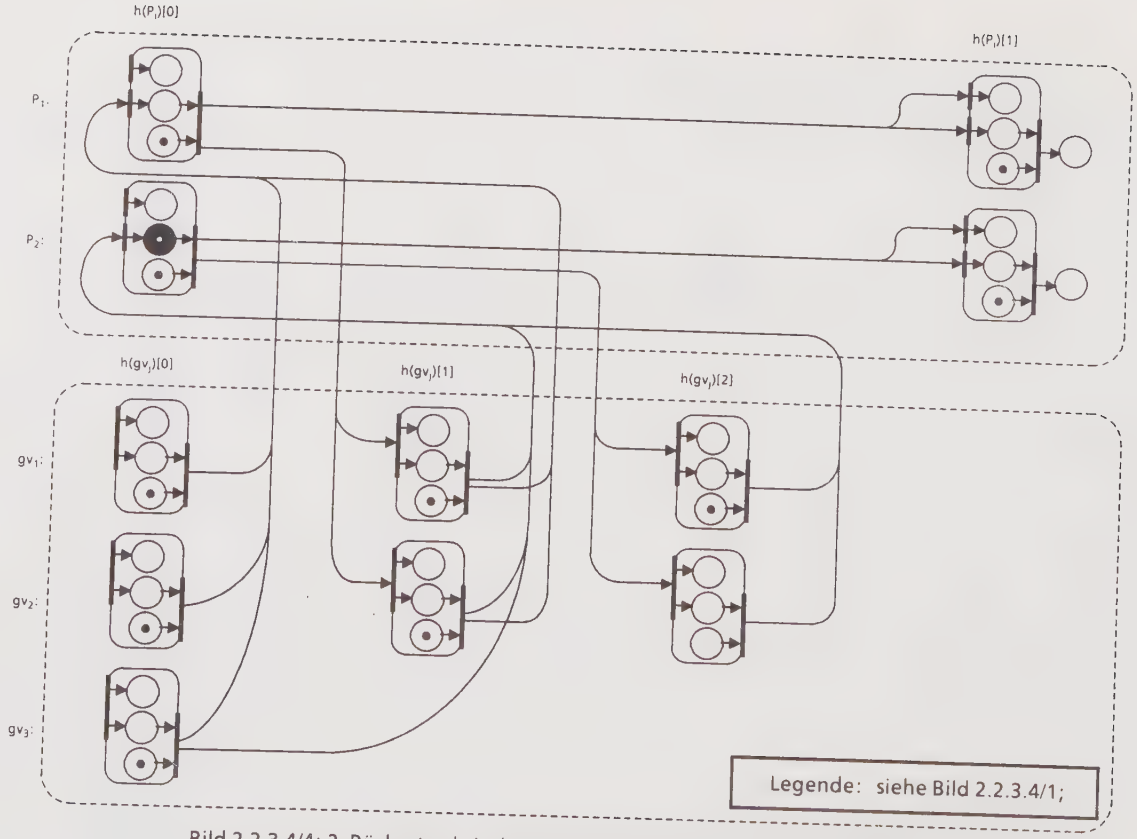


Bild 2.2.3.4/4: 2. Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

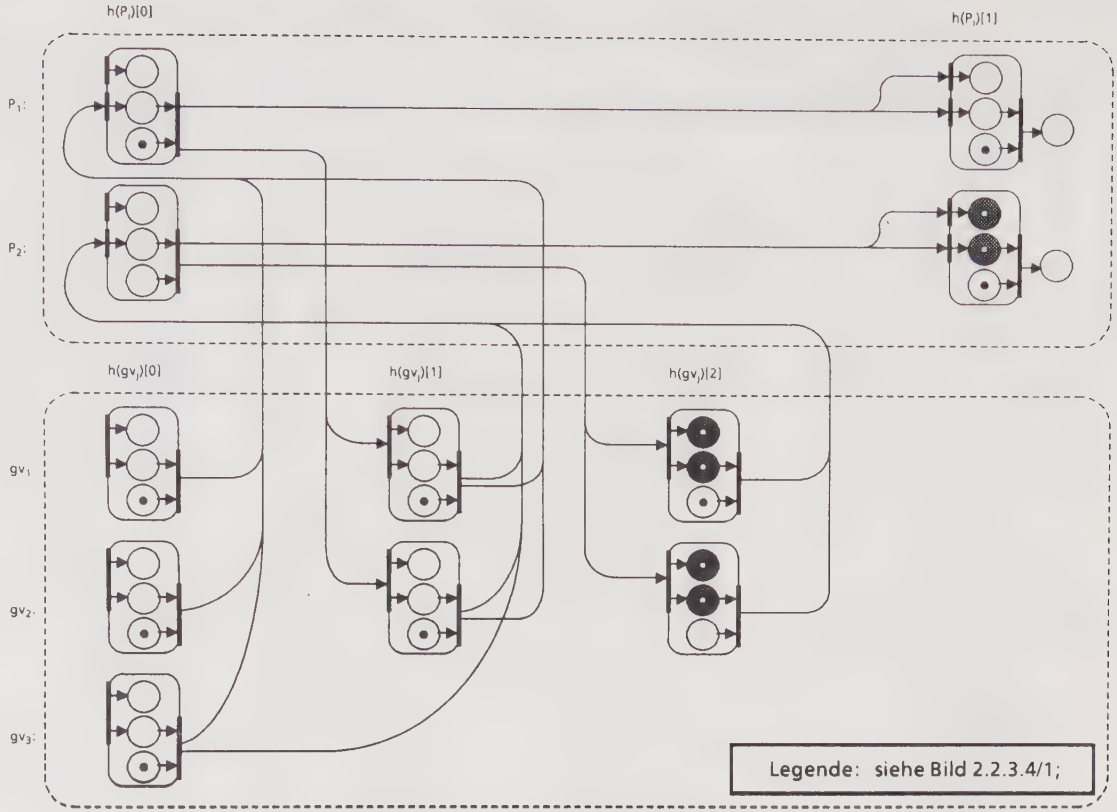


Bild 2.2.3.4/5: 3. Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

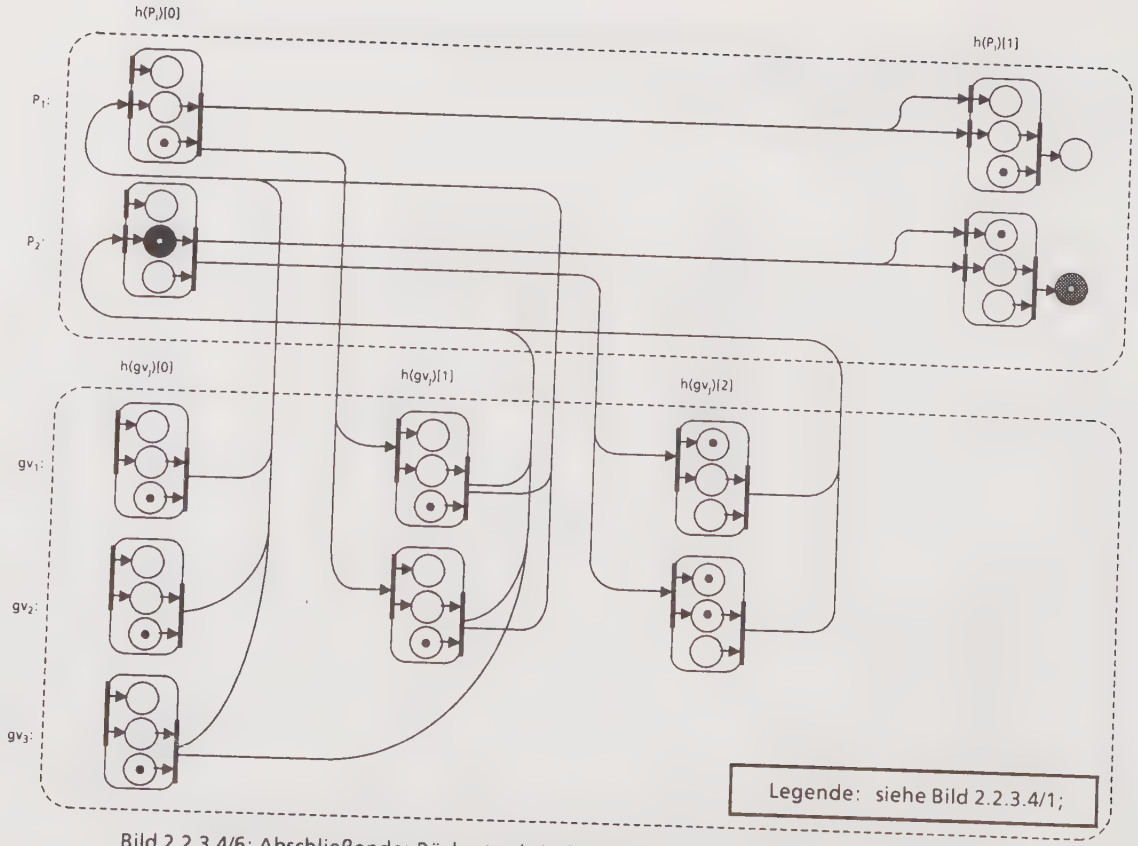


Bild 2.2.3.4/6: Abschließender Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

2.2.4. Berücksichtigung potentieller Fehlerverursacher

Mit den in den Bildern 2.2.3.4/3-6 skizzierten Rücksetzschritten ist es, basierend auf Algorithmus 2.2.3.2/1, möglich, nach einem aufgetretenen Rechensystemfehler einen im Sinne von Definition 2.2.3.1/2 konsistenten Prozeßsystem-Zustand wiederherzustellen. Unabhängig davon kann dieser jedoch im Sinne von Definition 2.2.3.2/1 fehlerhaft sein, denn Prozeßsystemfehler können mit den bisher eingeführten Mechanismen nicht erkannt und damit auch nicht gezielt beseitigt werden.
Auf welche Weise Prozeßsystemfehler aufgedeckt und inclusive ihrer potentiellen Ursachen beseitigt werden können, zeigen die folgenden Abschnitte.

2.2.4.1. Konzept der Akzeptanztests

Eine Grundvoraussetzung für die Behandlung von Prozeßsystemfehlern ist das Vorhandensein passender Konstrukte zu ihrer Erkennung und Lokalisierung. Da Prozeßsystemfehler als Abweichung des tatsächlichen Verhaltens vom spezifizierten zu betrachten sind, liegt es nahe zu ihrer Erkennung Tests zu verwenden, die auf einem Vergleich erwarteter Prozeßsystem-Zustände mit den errechneten beruhen. Im folgenden werden solche Tests als *Akzeptanztests* $AT_{i,k}$ bezeichnet.

Im vorliegenden Modell wird davon ausgegangen, daß mittels Akzeptanztests die Korrektheit von aktuellen Variablenwerten nachgewiesen werden kann.
Akzeptanztests sind auf einer Variablenmenge $V' \subseteq V_P$ eines Prozeßsystems $P=(A,D,w)$ definiert, welche sie, bezogen auf den während ihrer Ausführung bestehenden aktuellen Prozeßsystem-Zustand, verifizieren. Die Ausführung eines Akzeptanztests $AT_{i,k}$ erfolgt mittels einer Funktion *at*(..), die als Parameter den zu aktivierenden Akzeptanztest $AT_{i,k}$ und die mittels diesem zu verifizierende Variablenmenge V' enthält, sowie als Ergebnis den Wert **true**, für einen erfolgreich ausgeführten Akzeptanztest, und den Wert **false**, bei Erkennung eines/mehrerer Fehler, zurückliefert. Die Operation *at*(...) selbst ist Bestandteil einer atomaren Aktion, der Akzeptanztest ausführenden Aktion, von der im weiteren angenommen wird, daß sie die folgende Form hat:

Definition 2.2.4.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Eine Aktion $a_{i,j} \in A$ wird als eine den Akzeptanztest $AT_{i,k}$ auf der Variablenmenge $V' \subseteq V_P$ ausführende Aktion bezeichnet genau dann, wenn gilt:

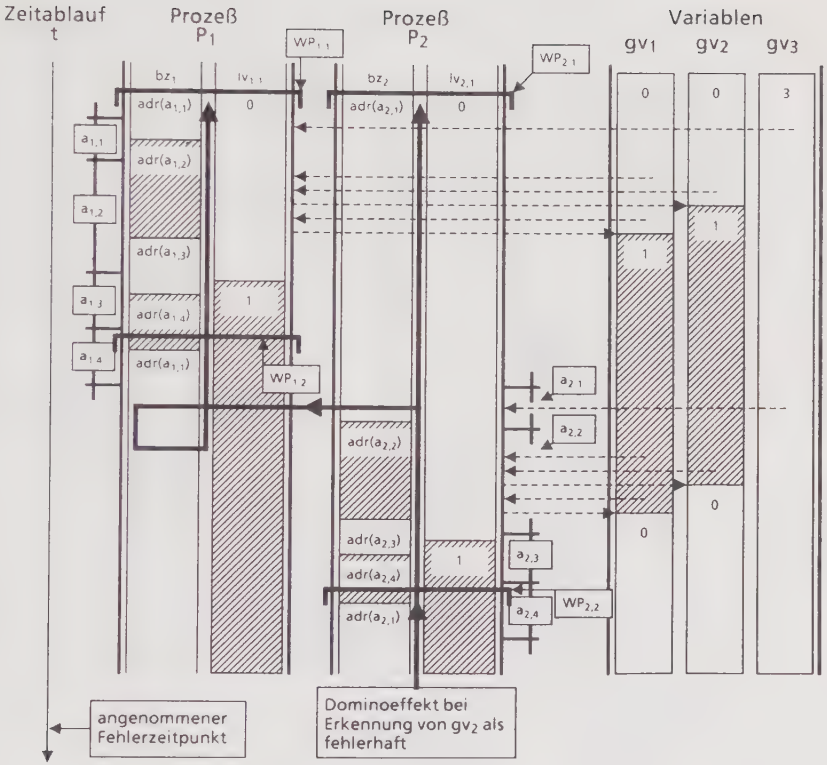
$a_{i,j}$: *when* $bz_i = adr(a_{i,j})$ *do if* $at(AT_{i,k}, V') =$ **true** *then* {...} *else* {...}; *od*;

Kurzschreibweise: $at(a_{i,j}) =$ **true** .<=> $a_{i,j}$ ist eine Akzeptanztest ausführende Aktion.

Bemerkung: Um ein Wiederauftreten eines erkannten Fehlers zu verhindern, kann es erforderlich sein, Aktionen, die Fehler verursacht haben könnten, von der weiteren Berechnung auszuschließen und durch Alternativen zu ersetzen. Hier bietet das *Recoveryblock-Konzept*, vgl. hierzu z.B. [Ran75], [Ran77], [Hor74], [HR73], Lösungen an. Das Recoveryblock-Konzept kann prinzipiell in das vorliegende Prozeßsystem-Modell und das gerade diskutierte Fehlerbehandlungs-Modell problemlos integriert werden. Es wird jedoch nicht weiter auf diese Möglichkeit eingegangen, da dies im Zusammenhang mit den bearbeiteten Fragestellungen ohne Bedeutung ist.

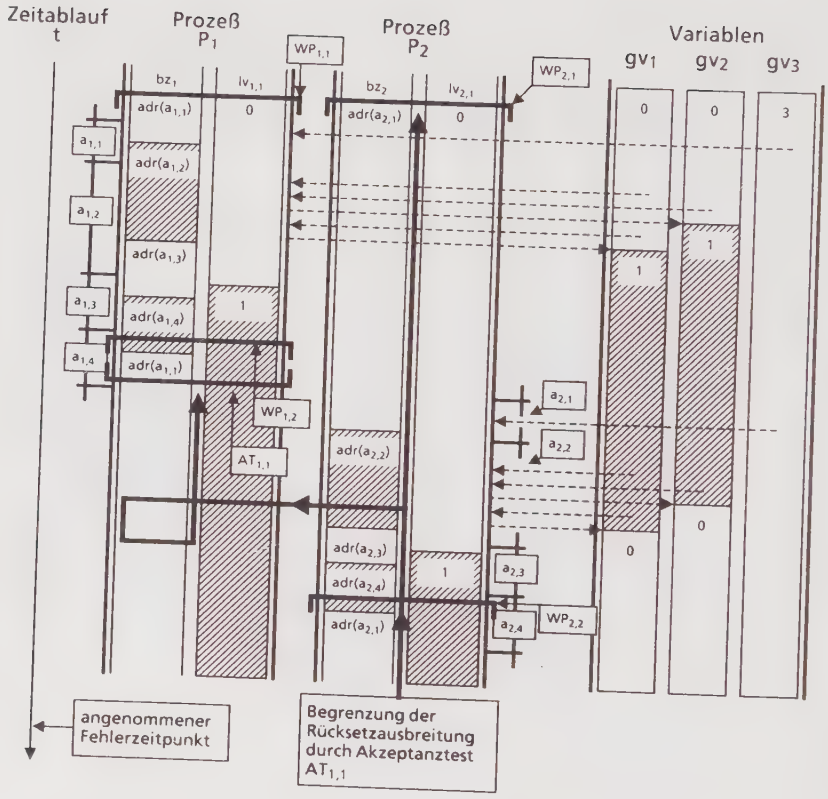
Bild 2.2.4.1/1 zeigt, daß im Falle der Berücksichtigung potentieller Fehlerverursacher in der Situation des Bildes 2.2.3.2/1, bei Erkennung von gv_2 als fehlerhaft, beispielsweise die Aktion $\alpha_{1,2}$ des Prozesses P_1 als potentieller Fehlerverursacher verdächtigt werden und Prozeß P_1 aus diesem Grunde zum Wiederaufsetzpunkt $WP_{1,1}$ zurückgesetzt werden müßte, um Änderungen von Aktion $\alpha_{1,2}$ rückgängig zu machen. Demgegenüber reicht in der gleichen Situation die Rücksetzung von P_2 zum Wiederaufsetzpunkt $WP_{2,1}$ zur Fehlerbeseitigung aus, falls die Korrektheit des von P_1 durch $\alpha_{1,2}$ modifizierten Variablenwertes von gv_2 in Akzeptanztest $AT_{1,1}$ nachgewiesen wurde (vgl. Bild 2.2.4.1/2).

Für das folgende wird vereinbart, daß ein erfolgreich verlaufener Nachweis der Korrektheit eines lokalen Prozeßsystem-Zustandes bzw. globalen Variablenwertes mittels einer sogenannten *Verifikationsanzeige*, die zu jedem lokalen Prozeßzustandswert (globalen Variablenwert) einer Zustands-Historie geführt wird, kenntlich gemacht wird und damit die Korrektheit eines gegebenen Zustandswertes/Variablenwertes durch Wiederanlaufprozeduren feststellbar ist.



Legende: siehe Bild 2.2.3.2/1.

Bild 2.2.4.1/1: Rücksetzausbreitung bei Berücksichtigung potentieller Fehlerverursacher



Legende: siehe Bild 2 2 3 2/1;

Bild 2.2.4.1/2: Begrenzung der Rücksetzausbreitung bei Berücksichtigung potentieller Fehlerverursacher durch Akzeptanztests

2.2.4.2. Fehlerursacherermittlungs- und -beseitigungsprozedur

Eine Wiederanlaufprozedur, welche bei der Wiederherstellung eines konsistenten Zustandes das Auftreten potentieller Fehlerverursacher berücksichtigt, stellt Algorithmus 2.2.4.2/1 dar

Algorithmus 2.2.4.2/1 unterscheidet sich von Algorithmus 2.2.3.2/1 nur bzgl. Schritt 2, in welchem durch Algorithmus 2.2.4.2/1 die Ermittlung der potentiellen Fehlerverursacher erfolgt. Die Schritte 1 und 3 bis 5 von Algorithmus 2.2.4.2/1 entsprechen den Schritten 1 bis 4 von Algorithmus 2.2.3.2/1 (vgl. Abschnitt 2.2.3.2), wobei bei Algorithmus 2.2.4.2/1 in Schritt 2 lediglich der Fall der Rücksetzung fehlerhafter, globaler Variablenwerte berücksichtigt ist, und in Schritt 2.4 eine Angleichung an Schritt 1 von Algorithmus 2.2.3.2/1 erfolgt.

Algorithmus 2.2.4.2/1 (Wiederanlaufprozedur bei Berücksichtigung potentieller Fehlerverursacher):

```
/* Algorithmus-BEGINN */

Schritt 0:  /* Variablen-Deklaration */

integer l, l1, l2;

for all Pi from P:    {struct element_of_h(Pi) h(Pi)[];};
for all gvk ∈ gVP:    {struct element_of_h(gv) h(gvk)[];};

set_of_struct ↑element_of_h(Pi) LWPR[];
set_of_struct ↑element_of_h(gv) LgSR[], LgLR;
set_of_struct ↑element_of_h(gv) LpFVR[];

Schritt 1:  /* Initialisierung; für alle Pj aus P und gvk ∈ gVP wird angenommen, daß
            h(Pj) bzw. h(gvk) die Wiederaufsetzpunkthistorie zu Pj bzw. gvk
            repräsentieren und bereits entsprechend initialisiert sind */

l := 0;

LWPR[0] := { ↑h(Pj)[k] |  ∃ h(gvm)[n]: der in h(gvm)[n] gesicherte Wert wurde als
                          fehlerhaft erkannt & h(gvm)[n].gSR = ↑h(Pj)[k]
           };

Schritt 2:  /* Ermittlung der potentiellen Fehlerverursacher */

do {

Schritt 2.1:    LpFVR[l] := { ↑h(gvm)[n] |  ∃ ↑h(Pj)[k] ∈ LWPR[l]: ↑h(gvm)[n] ∈ h(Pj)[k].LgLR
                                            &
                                            h(gvm)[n].gvm ist nicht verifiziert
                };
                l := l + 1;

Schritt 2.2:    LWPR[l] := { ↑h(Pj)[k] | ∃ ↑h(gvm)[n] ∈ LpFVR[l-1] & ↑h(Pj)[k] = h(gvm)[n].gSR};

Schritt 2.3:    LWPR[l] := LWPR[l-1] ∪ LWPR[l];

} while LWPR[l-1] ⊂ LWPR[l];
```

Schritt 2.4: /* **Mit den folgenden Zuweisungen wird erreicht, daß sich die** Wiederanlaufprozedur mit Berücksichtigung potentieller Fehlerverursacher **ab Schritt 3 nicht von derjenigen** ohne Berücksichtigung potentieller Fehlerverursacher **ab Schritt 2 unterscheidet** */

LWPR[0] : = LWPR[l] U { ↑ h(Pj)[k] l ∃ h(P_j)[m]: P_j muß zum Wiederaufsetzpunkt h(P_j)[m] zurückgesetzt werden & m ≤ k

||

∃ h(P_j)[m]: h(P_j)[m] ∈ LWPR[l] & m ≤ k

};

l : = 0;

Schritt 3: /* **Entspricht Schritt 2** der Wiederanlaufprozedur bei Nichtberücksichtigung potentieller Fehlerverursacher **(siehe Algoritmus 2.2.3.2/1)** */

Schritt 4: /* **Entspricht Schritt 3** der Wiederanlaufprozedur bei Nichtberücksichtigung potentieller Fehlerverursacher **(siehe Algoritmus 2.2.3.2/1)** */

Schritt 5: /* **Entspricht Schritt 4** der Wiederanlaufprozedur bei Nichtberücksichtigung potentieller Fehlerverursacher **(siehe Algoritmus 2.2.3.2/1)** */

/* **Algorithmus-ENDE** */

2.2.4.3. Verdeutlichung der Einzelschritte

Die Funktionsweise der durch Algorithmus 2.2.4.2/1 repräsentierten Fehlerursacherermittlungs- und -beseitigungsprozedur zeigen die beiden Beispiele 2.2.4.3/1 und 2.

Im Beispiel 2.2.4.3/1 werden die Einzelschritte, die zur Ermittlung eines konsistenten Wiederaufsetzzustandes in der Situation des Bildes 2.2.2.4/10 führen, skizziert, vgl. Bild 2.2.4.3/1. Dabei wird davon ausgegangen, daß keiner der in der Zustands-Historie abgespeicherten Werte als verifiziert gilt. Im Gegensatz dazu zeigt Beispiel 2.2.4.3/2, daß unter der Annahme der nachgewiesenen Korrektheit der Werte $h(P_1)[0]$, $h(P_2)[0]$, $h(gv_1)[0,1]$ $h(gv_2)[0,1]$ und $h(gv_3)[0]$ in der Situation von Bild 2.2.2.4/10 (vgl. Bild 2.2.4.3/2), Algorithmus 2.2.4.2/1 den erforderlichen Rücksetzaufwand zur Ermittlung eines konsistenten Wiederaufsetzzustandes wesentlich verringern kann. Dies ist in Bild 2.2.4.3/3 festgehalten.

Beispiel 2.2.4.3/1 (Im Zustand $val_b(a_{1,3}a_{1,4}a_{1,5}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ erforderliche Rücksetzausbreitung bei Berücksichtigung potentieller Fehlerverursacher und Erkennung des in $h(gv_2)[2]$ gesicherten Wertes als fehlerhaft):

Schritt 1:
```
LWPR[0]    =    {↑h(P2)[0]},
```

Schritt 2:
```
/*************************************************************************/
```

Schritt 2.1:
```
LpFVR[0]   :=   {↑h(gv1)[1], ↑h(gv2)[1], ↑h(gv3)[0]};
```

Schritt 2.2:
```
LWPR[1]    :=   {↑h(P1)[0]};
```

Schritt 2.3:
```
LWPR[1]    :=   {↑h(P1)[0], ↑h(P2)[0]},
```

Schritt 2.1:
```
LpFVR[1]   :=   {↑h(gv1)[1], ↑h(gv2)[1], ↑h(gv3)[0], ↑h(gv2)[0], ↑h(gv1)[0]},
```

Schritt 2.2:
```
LWPR[2]    :=   {↑h(P1)[0]};
```

Schritt 2.3:
```
LWPR[2]    :=   {↑h(P1)[0], ↑h(P2)[0]}    =    LWPR[1];
```

Schritt 2.4:
```
LWPR[0]    :=   {↑h(P1)[0], ↑h(P1)[1], ↑h(P2)[0], ↑h(P2)[1]};
```

Schritt 3:
```
/*************************************************************************/
```

Schritt 3.1:
```
LgSR[0]    :=   {↑h(gv1)[1], ↑h(gv2)[1], ↑h(gv1)[2], ↑h(gv2)[2]},
```

Schritt 3.2:
```
LWPR[1]    :=   {↑h(P2)[0]};
```

Schritt 3.3:
```
LWPR[1]    :=   {↑h(P1)[0], ↑h(P1)[1], ↑h(P2)[0], ↑h(P2)[1]}  =    LWPR[0];
```

Schritt 4:
```
/*************************************************************************/
```

Schritt 4.1:
```
LgLR       :=   {↑h(gv1)[0], ↑h(gv2)[0], ↑h(gv3)[0], ↑h(gv1)[1], ↑h(gv2)[1]};
```

Schritt 4.2:
```
h(gv1)[0].LgLR := "*";          h(gv1)[1].LgLR := "*";
h(gv2)[0].LgLR := "*";          h(gv2)[1].LgLR := "*";
h(gv3)[0].LgLR := "*";
```

Schritt 4.3:
```
h(gv1) := h(gv1)[0];
h(gv2) := h(gv2)[0];
h(gv3) := h(gv3)[0];
```

Schritt 4.4:
```
h(P1) := h(P1)[0];      h(P1)[0].LgSR := "*";      h(P1)[0].LgLR := "*",
h(P2) := h(P2)[0];      h(P2)[0].LgSR := "*";      h(P2)[0].LgLR := "*";
```

Schritt 5:
```
/*************************************************************************/
```

Schritt 5.1:
```
bz1 := adr(a1,1);    lv1,1 := 0;        bz2 := adr(a2,1);    lv2,1 := 0;
```

Schritt 5.2:
```
gv1 := 0;        gv2 := 0;        gv3 := 3;
```

Beispiel 2.2.4.3/2 (Falls die Verifikationsanzeige zu $h(gv_1)[1]$ und $h(gv_2)[1]$ auf **true** gesetzt wäre, vgl. Bild 2.2.4.3/2, wurde der Algorithmus 2.2.4.2/1 in der Situation von Beispiel 2.2.4.3/1 in den folgenden Schritten ablaufen):

Schritt 1: LWPR[0] := $\{\uparrow h(P_2)[0]\}$;

Schritt 2: /**********/

Schritt 2.1: LpFVR[0] := $\emptyset$;

Schritt 2.2: LWPR[1] := $\emptyset$;

Schritt 2.3: LWPR[1] := $\{\uparrow h(P_2)[0]\}$ = LWPR[0];

Schritt 2.4: LWPR[0] := $\{\uparrow h(P_2)[0], \uparrow h(P_2)[1]\}$;

/**********/

Schritt 3: Ab diesem Schritt stimmt das Ergebnis mit demjenigen überein, das man ab Schritt 3 bei Anwendung der Wiederanlaufprozedur ohne Berücksichtigung potentieller

Schritt 4: Fehlerverursacher im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ des Prozeßsystems P (vgl. Bild 2.2.2.4/10) im Beispiel 2.2.3.3/2 erhält.

Schritt 5: /**********/

Schritt 5.1: bz_2 := adr($a_{2,1}$); $lv_{2,1}$:= 0;

Schritt 5.2: gv_1 := 1; gv_2 := 1; gv_3 := 3;

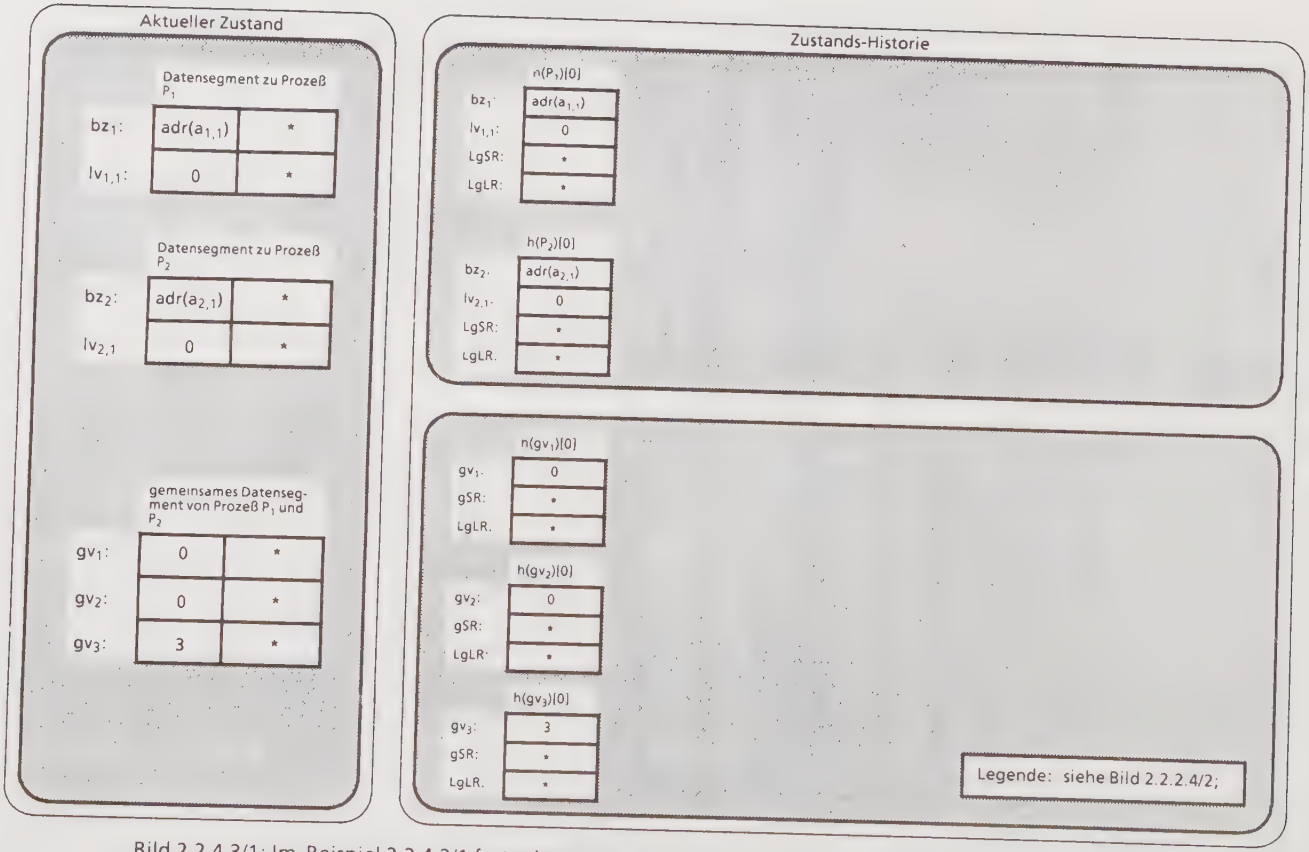


Bild 2.2.4.3/1: Im Beispiel 2.2.4.3/1 festgelegter Wiederaufsetzzustand (Beispiel-Prozeßsystem-Zustand $val_P(\lambda)$)

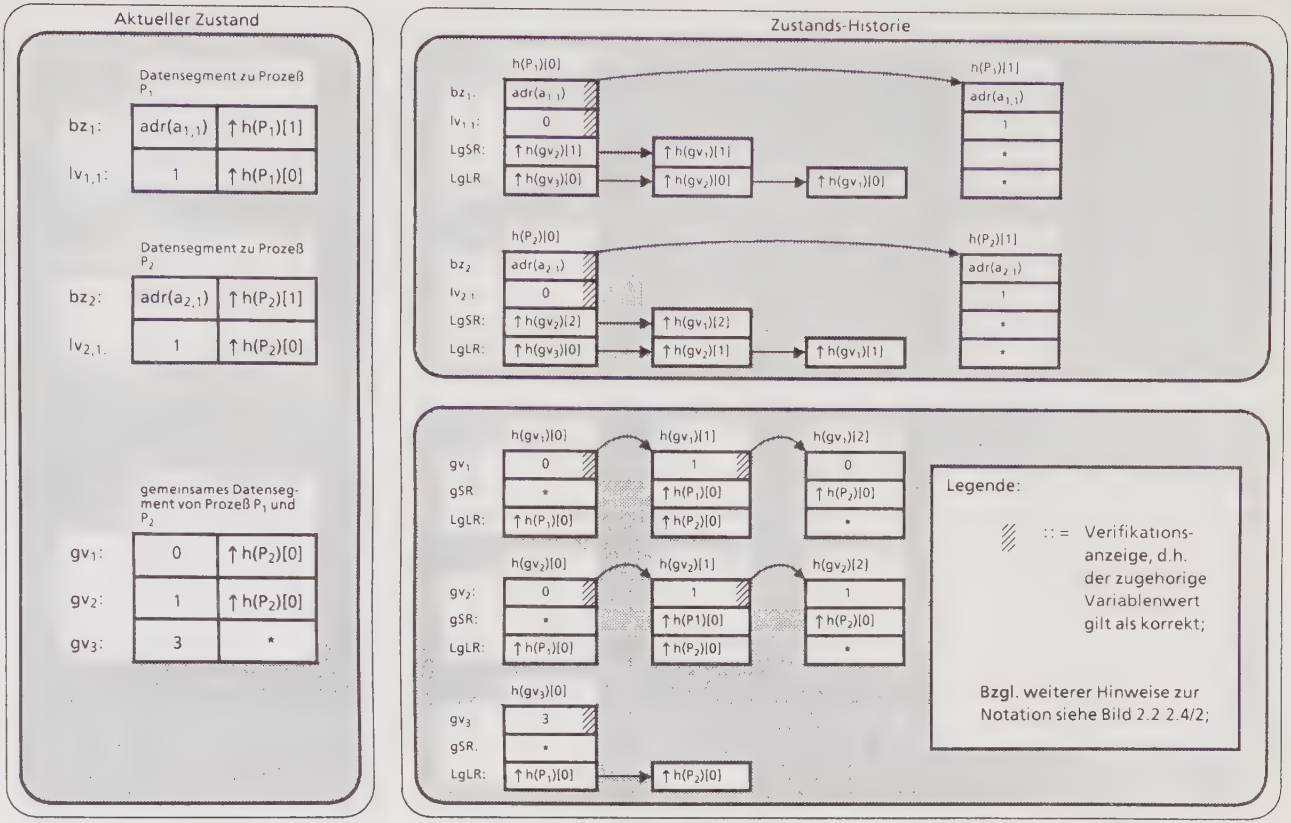


Bild 2.2.4.3/2: Beispiel-Prozeßsystem-Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ mit Verifikationsanzeige

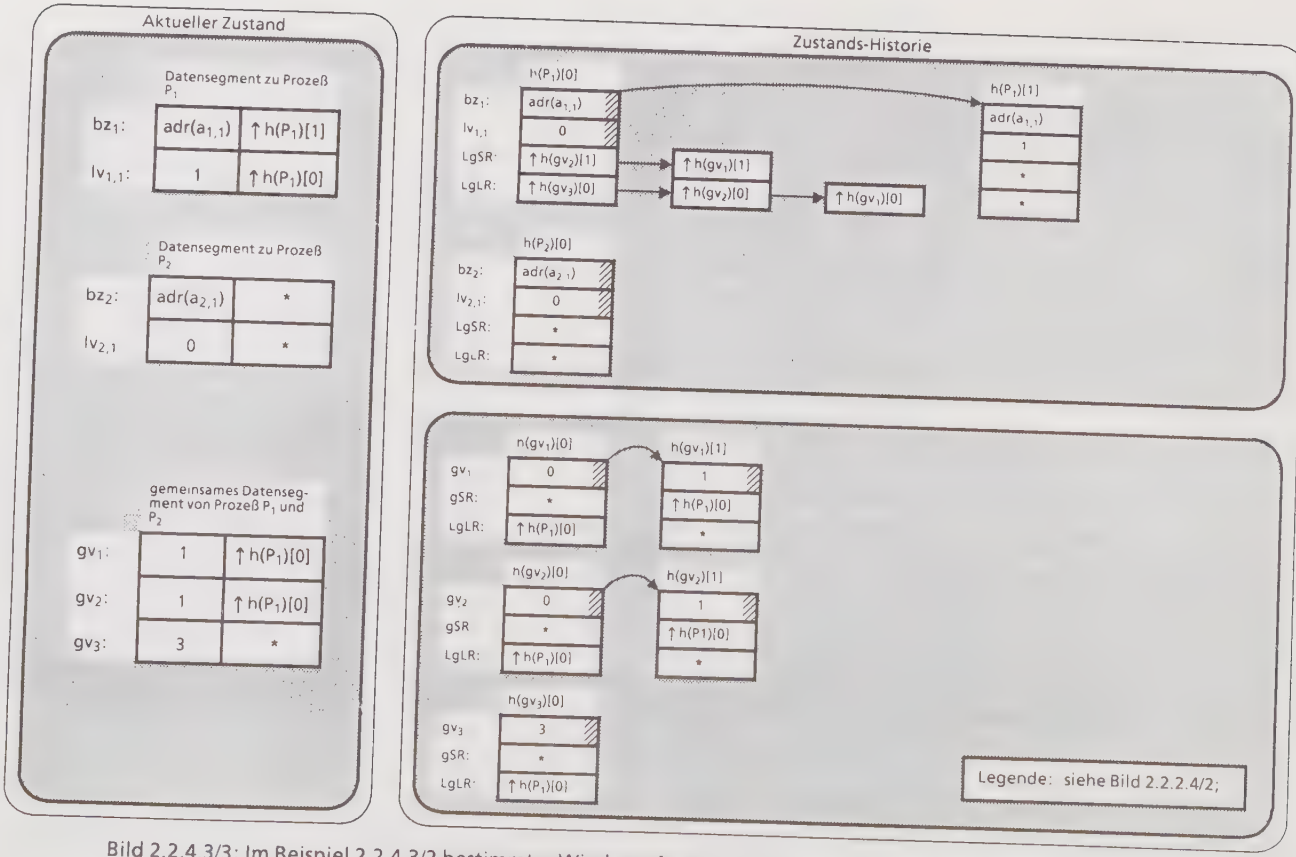


Bild 2.2.4.3/3: Im Beispiel 2.2.4.3/2 bestimmter Wiederaufsetzzustand (Beispiel-Prozeßsystem-Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4})$)

2.2.4.4. Veranschaulichung auf Petri-Netz-Basis

Auch im Falle von Algorithmus 2.2.4.2/1 erscheint eine Darstellung des Rücksetzvorganges mittels eines Petri-Netzes prinzipiell möglich und sinnvoll. Voraussetzung ist die Einbeziehung der Verifikationsanzeige in das verwendete Petri-Netz-Modell, inclusive der Existenz einer entsprechenden Vorschrift, wie in Abschnitt 2.2.3.4 erläutert, zur Abbildung der Zustands-Historie auf einen Petri-Netz-Graphen.

Analog zu Abschnitt 2.2.3.4, kann auch im Falle der Berücksichtigung potentieller Fehlerverursacher einer Zustands-Historie in folgender, eindeutiger Weise ein Petri-Netz zugeordnet werden:

(1) Jedes Element $h(P_i)[j]$ wird durch drei Stellen s_1, s_2, s_3 und drei Transitionen t_1, t_2, t_3 repräsentiert, eine exakte Definition von t_1, t_2 und t_3 gibt Tabelle 2.2.4.4/1 wieder, und

(2) jedem Element $h(gv_k)[l]$ werden fünf Stellen s_1 bis s_5 und vier Transitionen t_1 bis t_4 zugeordnet (vgl. Tabelle 2.2.4.4/2).

Die Modellierung der Verifikationsanzeige ist bei dieser Abbildungsvorschrift aus Gründen der Übersichtlichkeit nur für globale Variablen berücksichtigt. Eine Verallgemeinerung auf lokale Prozeßzustände ist ohne weiteres möglich.
Bzgl. der Notation und vereinfachten Darstellung gilt für das in Bild 2.2.4.4/1 zur Zustands-Historie des Bildes 2.2.2.4/10 konstruierte, strukturgleiche Petri-Netz analoges wie zu dem in Abschnitt 2.2.3.4 zu Bild 2.2.3.4/1 Gesagten.
Den der Zustands-Historie in Bild 2.2.2.4/10 zugeordneten Anfangszustand zeigt Bild 2.2.2.4/2. Die Markierung der Stellen $h(P_i)[j].s_3$, $h(gv_k)[l].s_1$ und $h(gv_k)[l].s_5$ mit einer Marke stellt sicher, daß die Transitionen $h(P_i)[j].t_3$, $h(gv_k)[l].t_3$ und $h(gv_k)[l].t_4$ nur maximal einmal schalten können. Dadurch kann wiederum als Endekriterium für die Modellierung eines Rücksetzvorganges die Nichtlebendigkeit des Petri-Netzes herangezogen werden.
Nach Erreichen des Endekriteriums ist der mit diesem verbundene Endzustand so zu interpretieren, daß alle Historienelemente $h(P_i)[j]$ und $h(gv_k)[l]$ zurückgesetzt werden müssen, für die gilt:

$$h(P_i)[j].s_1>0 \text{ bzw. } h(gv_k)[l].s_3>0.$$

Eine Markierung der Stellen $h(gv_k)[l].s_1$ (diese Stellen haben die Funktion der Verifikationsanzeige) zeigt eine noch nicht erfolgte Variablenverifizierung an. Aus diesem Grunde sind die Stellen $h(gv_1)[0,1].s_1$, $h(gv_2)[0,1].s_2$ und $h(gv_3)[0].s_1$ in dem zum Prozeßzustand des Bildes 2.2.4.3/2 konstruierten Petri-Netz, siehe Bild 2.2.4.4/3, nicht markiert.

Aktionsname	Φ	Θ
$h(P_i)[j].t_1, j>0$	$\Phi(h(P_i)[j-1].t_3)$	$h(P_i)[j].s_1 := h(Pi)[j].s_1 + 1;$
$h(P_i)[j].t_2$	$\bigcup_{\uparrow h(gv_k)[l] \in h(P_i)[j].LgLR} \Phi(h(gv_k)[l].t_3) \parallel$ $\bigcup_{\uparrow h(gv_k)[l] \in h(P_i)[j].LgSR} \Phi(h(gv_k)[l].t_3) \parallel$ $\bigcup_{\uparrow h(gv_k)[l] \in h(P_i)[j].LgLR} \Phi(h(gv_k)[l].t_4) \parallel$ $\bigcup_{\uparrow h(gv_k)[l] \in h(P_i)[j].LgSR} \Phi(h(gv_k)[l].t_4) \parallel$ $\Phi(h(P_i)[j-1].t_3)$	$h(P_i)[j].s_2 := h(Pi)[j].s_2 + 1;$
$h(P_i)[j].t_3$	$h(P_i)[j].s_2>0$ & $h(P_i)[j].s_3>0$	$\forall \uparrow h(gv_k)[l] \in h(P_i)[j].LgLR: \Theta(h(gv_k)[l].t_1);$ $\forall \uparrow h(gv_k)[l] \in h(P_i)[j].LgSR: \Theta(h(gv_k)[l].t_2);$ $\Theta(h(P_i)[j+1].t_1);$ $\Theta(h(P_i)[j+1].t_2);$ $h(P_i)[j].s_2 := h(Pi)[j].s_2 - 1;$ $h(P_i)[j].s_3 := h(Pi)[j].s_3 - 1;$

Tabelle 2.2.4.4/1: h(Pi)[j] charakterisierende Transitionen t_1, t_2 und t_3

Aktionsname	Φ	Θ
$h(gv_k)[l].t_1$	$\bigcup_{\uparrow h(P_i)[j] \in h(gv_k)[l].LgLR} \Phi(h(P_i)[j].t_3)$	$h(gv_k)[l].s_2 := h(gv_k)[l].s_2 + 1;$
$h(gv_k)[l].t_2, l>0$	$h(gv_k)[l].gSR = \uparrow h(P_i)[j]: \Phi(h(P_i)[j].t_3)$	$h(gv_k)[l].s_4 := h(gv_k)[l].s_4 + 1;$
$h(gv_k)[l].t_3$	$h(gv_k)[l].s_1>0$ & $h(gv_k)[l].s_2>0$	$\forall \uparrow h(P_i)[j] \in h(gv_k)[l].LgLR: \Theta(h(P_i)[j].t_2);$ $h(gv_k)[l].gSR = \uparrow h(P_i)[j]: \Theta(h(P_i)[j].t_2);$ $h(gv_k)[l].s_1 := h(gv_k)[l].s_1 - 1;$ $h(gv_k)[l].s_2 := h(gv_k)[l].s_2 - 1;$ $h(gv_k)[l].s_3 := h(gv_k)[l].s_3 + 1;$
$h(gv_k)[l].t_4$	$h(gv_k)[l].s_4>0$ & $h(gv_k)[l].s_5>0$	$\forall \uparrow h(P_i)[j] \in h(gv_k)[l].LgLR: \Theta(h(P_i)[j].t_2);$ $h(gv_k)[l].gSR = \uparrow h(P_i)[j]: \Theta(h(P_i)[j].t_2);$ $h(gv_k)[l].s_3 := h(gv_k)[l].s_3 + 1;$ $h(gv_k)[l].s_4 := h(gv_k)[l].s_4 - 1;$ $h(gv_k)[l].s_5 := h(gv_k)[l].s_5 - 1;$

Tabelle 2.2.4.4/2: $h(gv_k)[l]$ darstellende Transitionen t_1, t_2, t_3 und t_4

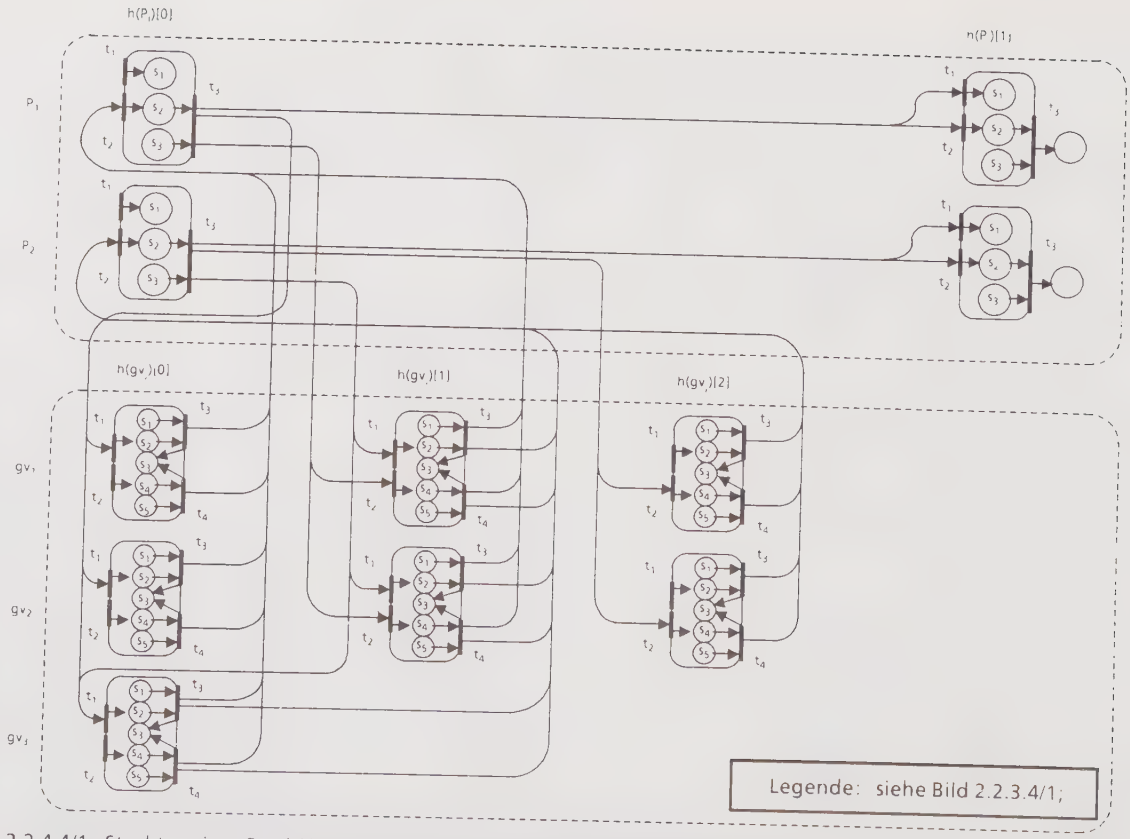


Bild 2.2.4.4/1: Struktur eines Petri-Netzes zur Darstellung der Zustands-Historie von Bild 2.2.4.3/2 mit Verifikationsanzeige für globale Variablenwerte

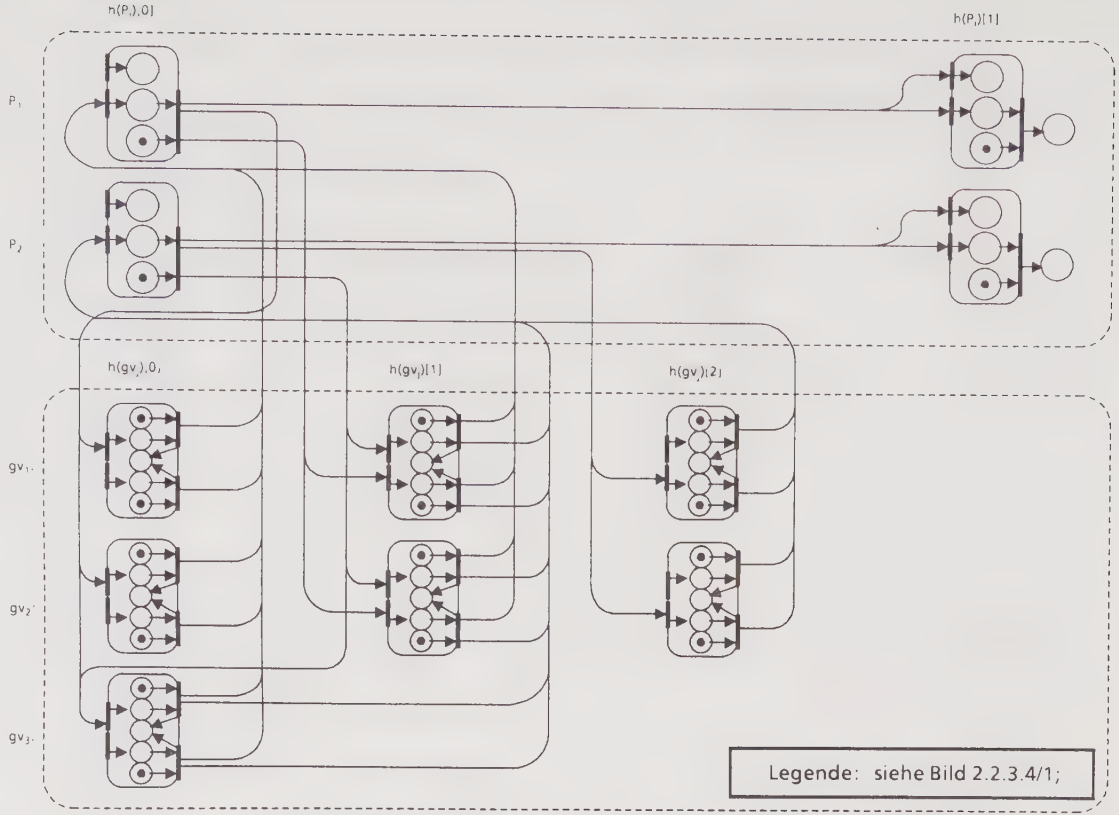


Bild 2.2.4.4/2: Petri-Netz-Darstellung der Zustands-Historie von Bild 2.2.2.4/10

Bild 2.2.4.4/3: Petri-Netz-Darstellung der Zustands-Historie von Bild 2.2.4.3/2

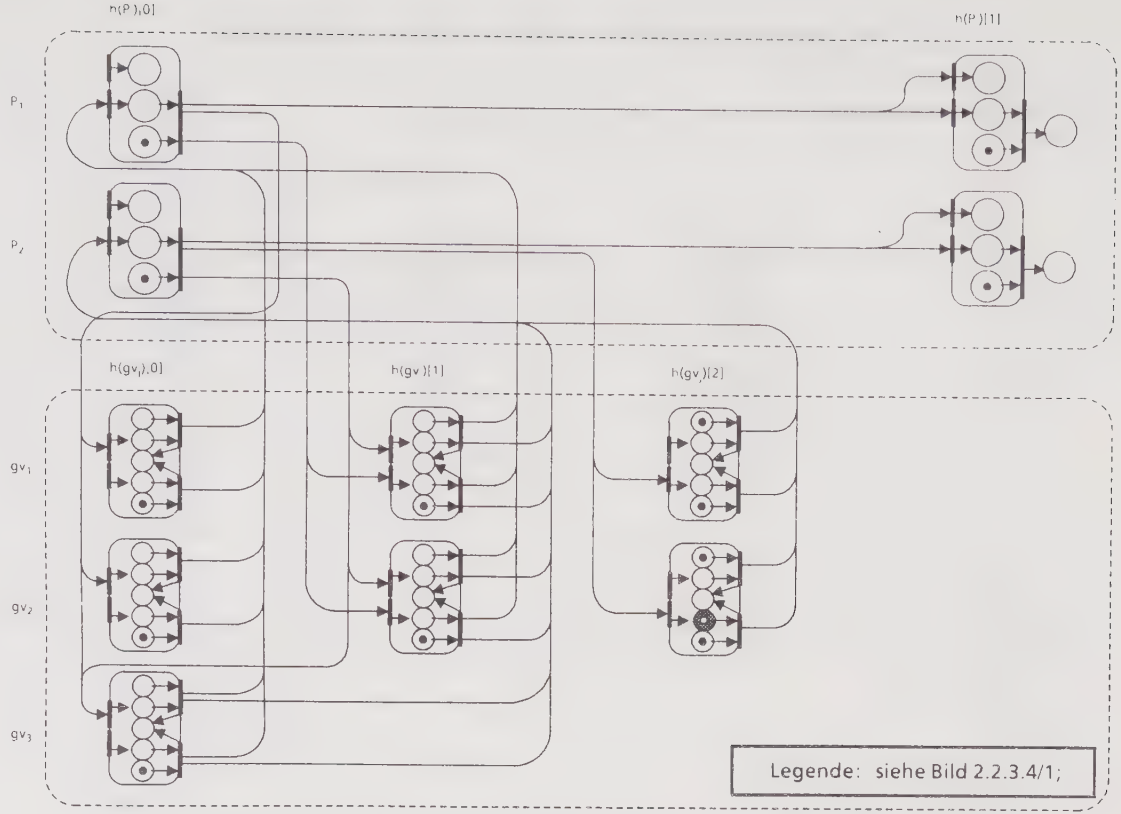


Bild 2.2.4.4/4: 1. Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

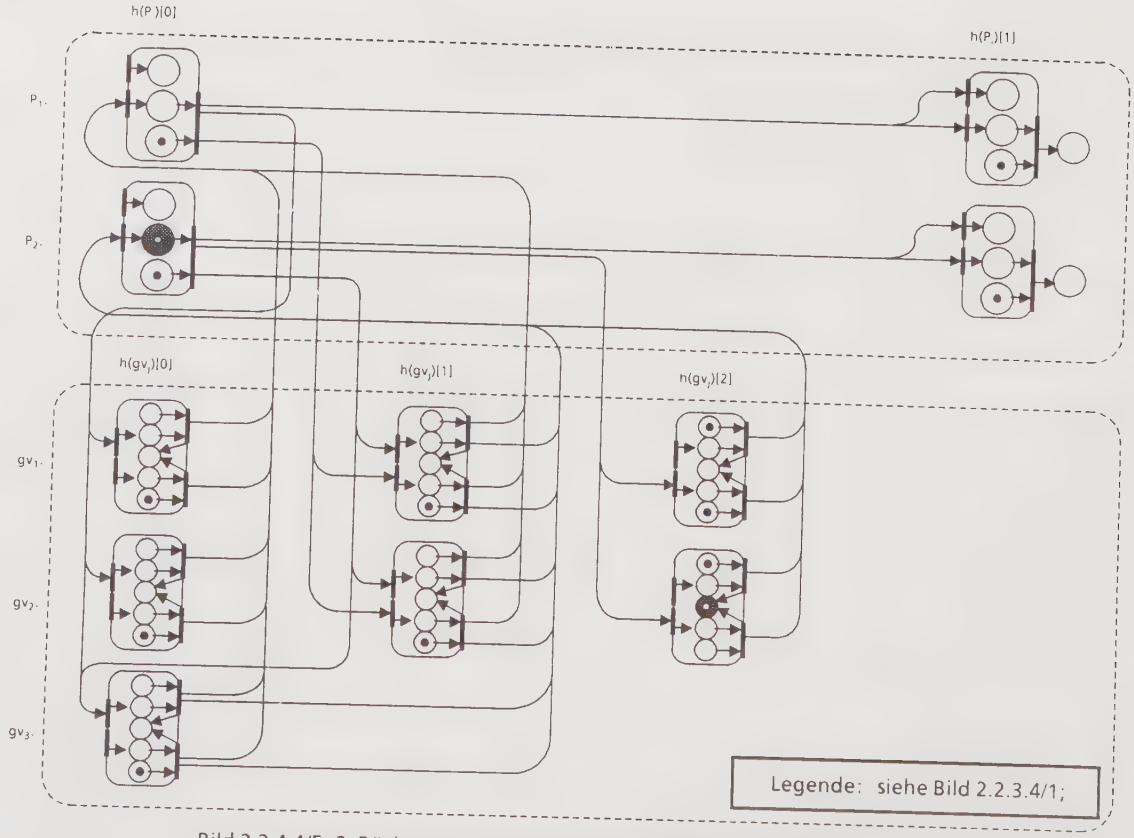


Bild 2.2.4.4/5: 2. Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

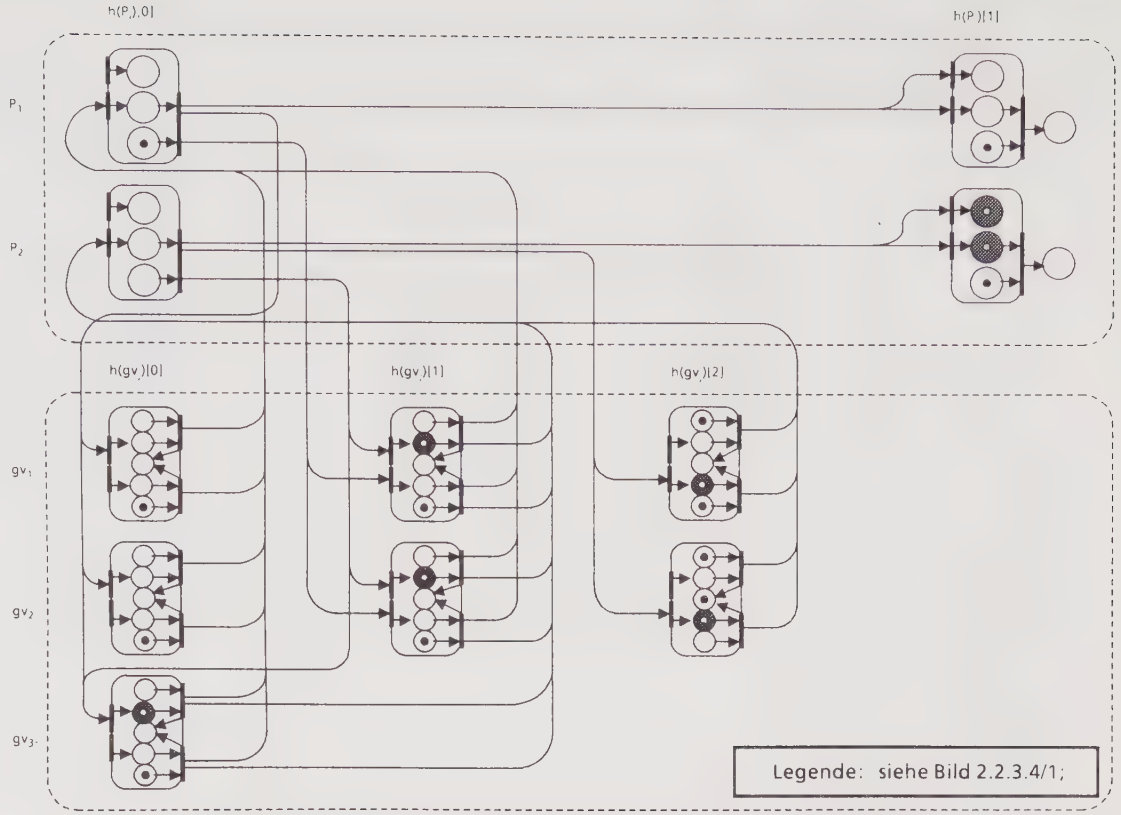


Bild 2.2.4.4/6: 3. Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

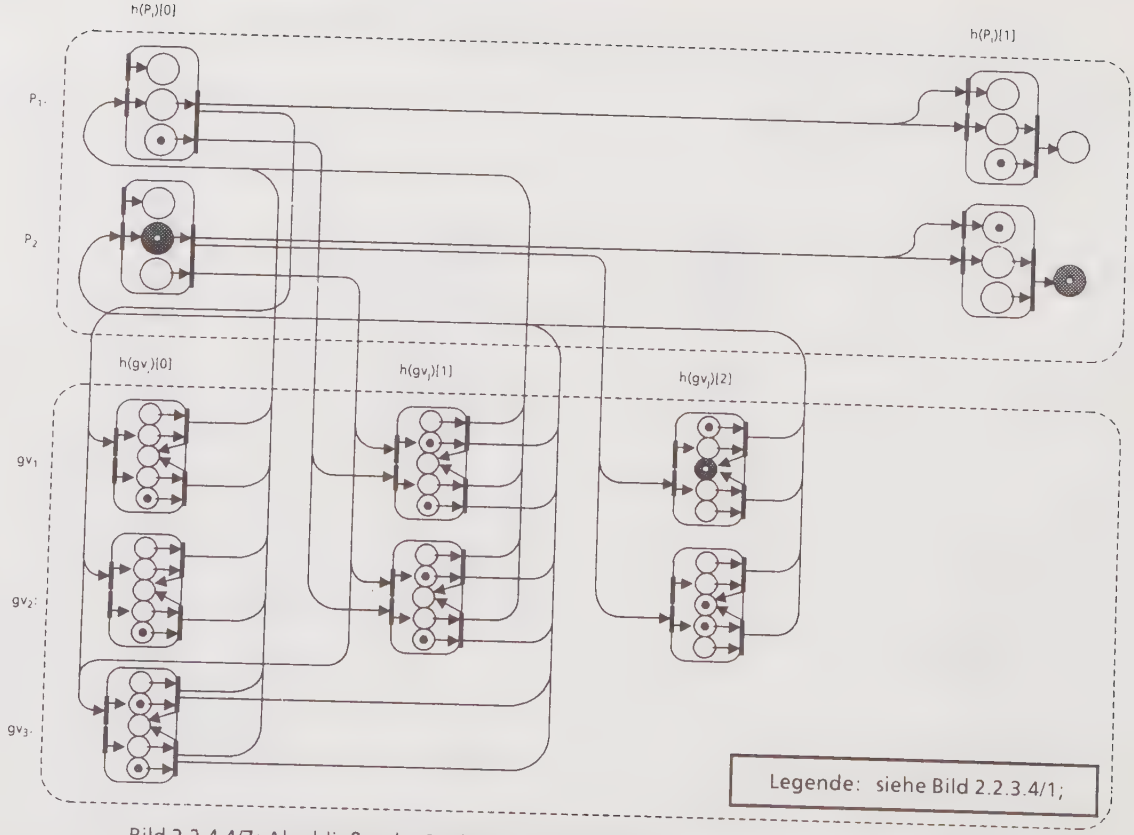


Bild 2.2.4.4/7: Abschließender Rücksetzschritt bei Erkennung von $h(gv_2)[2]$ als fehlerhaft

Im Vergleich zu dem in Abschnitt 2.2.3.4 skizzierten Rücksetzvorgang ist bei der Interpretation der in den Bildern 2.2.4.4/4-7 dargestellten Fehlerbehandlungsschritte zusätzlich zu berücksichtigen, daß die Korrektheit von Variablenwerten auch durch sie nicht erzeugende, aber lesende Aktionen/Prozesse (in Bild 2.2.4.4/4 z.B. für Prozeß P_2 durch die Verbindung $h(P_2)[0].t_3 \rightarrow h(gv_1)[1].t_1$ zum Ausdruck gebracht) in Frage gestellt werden kann

Der Wunsch nach Rücksetzung des in $h(gv_2)[2]$ gespeicherten Variablenwertes wird in Bild 2.2.4.4/4 durch Markierung der Stelle $h(gv_2)[2].s_4$ verdeutlicht.

3. Recoveryfunktionskonzept

Basierend auf dem in Abschnitt 2.1 eingeführten Prozeßsystem-Modell läßt sich das in 2.2 beschriebene Fehlerbehandlungskonzept durch Einführung des Begriffes einer *Recoveryfunktion* (*Fehlerbehandlungsfunktion*) weiter formalisieren. Es läßt sich zeigen, daß - abhängig von der Prozeßsystemklasse, der eine Berechnung zuzuordnen ist - im Hinblick auf das Kriterium *Minimierung des Fehlerbehandlungsaufwandes*, gemessen als Zahl der zurückzusetzenden Aktionen, in einer gegebenen Fehlersituation verschiedene Typen von Recoveryfunktionen zu unterscheiden sind.
Neben der Beschreibung dreier verschiedener Recoveryfunktionen im letzten Abschnitt dieses Kapitels, werden zuvor in drei Abschnitten alle dafür wesentlichen Vereinbarungen, Definitionen, Annahmen und Voraussetzungen, die nicht Gegenstand der Modellbildung in Kapitel zwei waren, zusammengestellt.

Im einzelnen werden zunächst durch Abstraktion vom vorgestellten Fehlerbehandlungskonzept in Abschnitt 3.1 Begriffe abgeleitet, die im weiteren Verwendung finden werden. Darauf aufbauend wird in Abschnitt 3.2 die Menge zurückzusetzender Aktionen beschrieben, und in Abschnitt 3.3 werden zu unterscheidende Prozeßsystem-Zustände definiert.

3.1. Abstraktion vom Fehlerbehandlungskonzept

Die Abstraktion vom in Abschnitt 2.2 vorgestellten Fehlerbehandlungskonzept führt in diesem Abschnitt zur Formalisierung der wichtigen Bezeichnungen *Endezeitpunkt einer Aktion*, *aktuellster Wiederaufsetzpunkt zu einer Aktion*, *Recoveryeinheit* und *Einheitsrelation*. In den folgenden vier Abschnitten 3.1.1 bis 3.1.4 werden die genannten Begriffe eingeführt, motiviert und anhand von Beispielen, in welchen auf die Bilder des Kapitels zwei bezug genommen wird, erläutert.

3.1.1. Endezeitpunkt einer Aktion

Die Präzisierung des Abschlußzeitpunktes einer Aktion $\alpha[i]$ einer aktiven Berechnungsfolge α relativ zu den anderen Aktionen von α steht bei den folgenden Ausführungen im Mittelpunkt. Es wird gezeigt, daß je zwei Aktionen $\alpha[i]$ und $\alpha[j]$ einer Berechnungsfolge α bzgl. ihres zeitlichen Auftretens in α mittels der Relation $<$ vergleichbar sind. Dies ist u.a. auf die Annahme nach der zeitlichen Unteilbarkeit von Aktionen (vgl. hierzu die Bemerkung am Ende von Abschnitt 2.1.1.1) zurückzuführen. Diese Eigenschaft wird, wie bereits in Abschnitt 2.1 erwähnt, als *Atomarität* bezeichnet und im zweiten Teil dieses Abschnittes vertieft.

Definition 3.1.1/1: Sei α ein aktiver Ablauf eines Prozeßsystems $P=(A,D,w)$. $\alpha[i]$ sei eine Aktion der Folge α. Dann bezeichnet $t(\alpha[i])$ den *Endezeitpunkt der Aktion* $\alpha[i]$, d.h. den Zeitpunkt, zu dem die Aktion $\alpha[i]$ bei der Ausführung der Folge α vollständig abgeschlossen wurde.

Hilfssatz 3.1.1/1: Sei α eine aktive Berechnung eines Prozeßsystems $P=(A,D,w)$. $\alpha[i]$, $\alpha[j]$ seien Aktionen von α. Ist $i \neq j$, so folgt:

$$t(\alpha[i]) < t(\alpha[j]) \parallel t(\alpha[j]) < t(\alpha[i]).$$

Beweis: Dies folgt unmittelbar aufgrund der Annahme der Atomarität von Aktionen, die sich in der Definition der Funktion val_P und damit in der Berechnung des Zustandes $val_P(\alpha)$ zu einem aktiven Ablauf α widerspiegelt. ∎

Der Hilfssatz 3.1.1/1 stellt für das Weitere sicher, daß zwei verschiedene Aktionen einer aktiven Berechnungsfolge bzgl. der zeitlichen Relation < vergleichbar sind, d.h. Aktionen können entsprechend ihrem Endezeitpunkt geordnet werden. Da die zeitliche Vergleichbarkeit für die Endezeitpunkte zweier beliebiger Aktionen einer Folge α gilt, ist diese zeitliche Ordnung bzgl. der Aktionen aus $W(\alpha)$ vollständig, d.h. sowohl transitiv, asymmetrisch als auch zusammenhängend. Dies bedeutet jedoch nicht, daß in einem Mehrrechnersystem oder Mehrprozessorsystem die Ausführung von Aktionen tatsächlich sequentiell erfolgen muß und die Endezeitpunkte zweier Aktionen einer aktiven Berechnungsfolge zeitlich nicht zusammenfallen dürfen. Es muß lediglich sichergestellt sein, daß jedes Berechnungsergebnis sequentiell errechnet werden kann. Dadurch wird die Zulässigkeit des impliziten Rückgriffs auf die Existenz eines derartigen Ablaufs in den folgenden Beweisen dieser Arbeit gewährleistet.

Annahme 3.1.1/1: Bzgl. der Endezeitpunkte von Aktionen $\alpha[i]$, $\alpha[j]$ einer aktiven Berechnungsfolge α eines Prozeßsystems $P=(A,D,w)$ wird im folgenden angenommen, daß gilt:

$$t(\alpha[i]) \leq t(\alpha[j]) \iff i \leq j.$$

Auf Annahme 3.1.1/1 wurde bereits in Definition 2.1.3.1/1 implizit zurückgegriffen. Sie ist auf der Basis der Definition der Funktion val_P (vgl. Abschnitt 2.1.1.3) als im zugrundegelegten Prozeßsystem-Modell gültig ableitbar. Auf einen expliziten Beweis dieses Sachverhalts wird an dieser Stelle verzichtet.

Einer weiter vertiefenden Erläuterung der Eigenschaft einer Aktion atomar zu sein dienen die abschließenden Ausführungen dieses Abschnittes.

Definition 3.1.1/2: *Wertanderungen* einer Aktion $\alpha[i]$ auf einer Variablenmenge $V_{\alpha[i]}$ bezeichnet man als *atomar* genau dann, wenn diese Anderungen für andere Aktionen erst sichtbar werden, wenn sie bzgl *aller* Variablen von $V_{\alpha[i]}$ vollständig abgeschlossen sind

Definition 3.1.1/3: Eine Aktion $\alpha[i]$ wird als *atomare Aktion* bezeichnet, wenn alle ihre Wertänderungen atomar sind.

Annahme 3.1.1/2: Alle im folgenden betrachteten Aktionen werden als atomar angenommen.

Bemerkung: Gehören die Aktionen $\alpha[i]$ und $\alpha[j]$, $i \neq j$, demselben Prozeß an, dann ist $\alpha[i]$ bzw $\alpha[j]$ aus Sicht von $\alpha[j]$ bzw $\alpha[i]$ in einem Prozeßsystem $P=(A,D,w)$ atomar. Die Atomarität von Aktionen desselben Prozesses zueinander wird über die implizite Synchronisation dieser Aktionen mittels des gemeinsamen Befehlszählers erreicht. Zwischen Aktionen, die nicht zum gleichen Prozeß gehören, muß die Forderung nach Atomarität durch explizite Synchronisation erzwungen werden. Der Betriebssystemkern muß dafür entsprechende Mechanismen bereitstellen.

Beispiel 3.1.1/1: Für die Aktionen des in Bild 2.2.3.2/1 dargestellten aktiven Berechnungsablaufs α des Beispiel-Prozeßsystems P (siehe Bild 2.2.2.4/1) gilt:

$$t(a_{1,1})<t(a_{1,2})<t(a_{1,3})<t(a_{1,4})<t(a_{2,1})<t(a_{2,2})<t(a_{2,3})<t(a_{2,4}).$$

3.1.2. Aktuellster Wiederaufsetzpunkt

Unter Verwendung des Begriffes Endezeitpunkt einer Aktion läßt sich der *aktuellste Wiederaufsetzpunkt* zu einer gegebenen, innerhalb einer aktiven Folge vorkommenden Aktion wie folgt beschreiben und errechnen.

Definition 3.1.2/1: Sei α ein aktiver Berechnungsablauf eines Prozeßsystems $P=(A,D,w)$ und $\alpha[i]$ eine Aktion von α. Dann bezeichnet $WP_\alpha(\alpha[i])$ den *aktuellsten Wiederaufsetzpunkt* bzgl des Prozesses $P_{bz(\alpha[i])}$, der vor dem Endezeitpunkt $t(\alpha[i])$ der Aktion $\alpha[i]$ gesetzt wurde.

Der aktuellste Wiederaufsetzpunkt $WP_\alpha(\alpha[i])$ zu einer Aktion $\alpha[i]$ einer aktiven Berechnungsfolge α bezeichnet denjenigen Wiederaufsetzpunkt, zu dem die Berechnung α, bezogen auf den Prozeß $P_{bz(\alpha[i])}$, zumindest zurückgesetzt werden muß, falls $\alpha[i]$ fehlerhaft ist oder aufgrund von Abhängigkeiten zu anderen Aktionen, deren

Korrektheit in Frage gestellt wird, einer Fehlerbehandlung unterworfen werden muß.

Definition 3.1.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnung von P und $\alpha[i]$ eine Aktion von α. Dann ist der aktuellste Wiederaufsetzpunkt $WP(\alpha[i])$ zur Aktion $\alpha[i]$ wie folgt definiert:

$$WP_\alpha(\alpha[i]) ::= \begin{cases} WP_{\alpha[j]} & wp(\alpha[j])=\textbf{true} \;\&\; proc_P(\alpha[i],\alpha[j]) \;\&\; \neg\exists\alpha[k]\,[0\le j<k<i \;\&\; wp(\alpha[k])=\textbf{true} \;\&\; proc_P(\alpha[k],\alpha[j])] \\ \qquad\qquad falls & \\ WP_\lambda|lV_{P_{bez(\alpha[i])}} & \neg\exists\alpha[j]\,[0\le j<i \;\&\; wp(\alpha[j])=\textbf{true} \;\&\; proc_P(\alpha[j],\alpha[i])] \end{cases}$$

dabei wird mit $WP_\lambda|lV_{P_{bez(\alpha[i])}}$ der *Initialisierungszustand* eingeschränkt auf die lokalen Variablen des Prozesses $P_{bez(\alpha[i])}$ bezeichnet; $WP_{\alpha[j]}$ ist der durch $\alpha[j]$ erzeugte Wiederaufsetzpunkt.
Zur Erinnerung: $wp(\alpha[j])=\textbf{true}$ gilt genau dann, wenn $\alpha[j]$ eine Wiederaufsetzpunktaktion ist.

Beispiel 3.1.2/1: Der aktuellste Wiederaufsetzpunkt der Aktionen $a_{i,j}$, $i\in\{1,2\}$, $j\in\{1,2,3,4\}$, des in Bild 2.2.3.2/1 wiedergegebenen Berechnungsablaufs α ist:

$$WP_\alpha(a_{i,j}) = WP_\lambda|lV_{P_{bez(a_{i,j})}} = WP_{i,1}.$$

Mit $WP_{i,1}$ wurde der Wiederaufsetzpunkt $WP_\lambda|lV_{P_{bez(a_{i,j})}}$ in Bild 2.2.3.2/1 bezeichnet.

3.1.3. Begriff der Recoveryeinheit

Eng verbunden mit der Bezeichnung aktuellster Wiederaufsetzpunkt zu einer Aktion ist der Begriff der *Recoveryeinheit* (*Fehlerbehandlungseinheit*). Dieser bezieht sich auf Gruppen von Aktionen.
Aktionen, die eine Recoveryeinheit bilden, sind, bei Zugrundelegung der in den Abschnitten 2.2.3.2 und 2.2.4.2 vorgestellten Wiederanlaufverfahren, voneinander (fehlerbehandlungstechnisch) abhängig.

Damit zwei Aktionen einer Berechnungsfolge α als zur gleichen Recoveryeinheit gehörend betrachtet werden können, müssen sie demselben Prozeß zugeordnet sein; zum anderen muß ihr aktuellster Wiederaufsetzpunkt identisch sein.

Definition 3.1.3/1: Sei $P=(A,D,w)$ ein Prozeßsystem und α eine aktive Berechnungsfolge von P. $\alpha[i]$ und $\alpha[j]$ seien zwei Aktionen der Folge α. $\alpha[i]$ und $\alpha[j]$ bilden eine *Recoveryeinheit* genau dann, wenn gilt:

$$proc_P(\alpha[i],\alpha[j]) \;\&\; WP_\alpha(\alpha[i]) = WP_\alpha(\alpha[j]).$$

Beispiel 3.1.3/1: Im aktiven Berechnungsablauf $\alpha = a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4}$ des Beispiel-Prozeßsystems P des Bildes 2.2.2.4/1 bilden die vier Aktionen $a_{i,1}$, $a_{i,2}$, $a_{i,3}$ und $a_{i,4}$, $i\in\{1,2\}$, des Prozesses P_1 bzw. P_2 jeweils eine Recoveryeinheit.

Hilfssatz 3.1.3/1: Jede Aktion $\alpha[i]$ einer aktiven Berechnung α eines Prozeßsystems $P=(A,D,w)$ gehört genau einer Recoveryeinheit an.

Beweis: Nach Definition 3.1.2/2 besitzt jede Aktion einer aktiven Berechnungsfolge einen eindeutig definierten aktuellsten Wiederaufsetzpunkt. Damit gehört sie, unter Rückgriff auf Definition 3.1.3/1, auch zu genau einer Recoveryeinheit. ■

3.1.4. Einheitsrelation

Formal läßt sich der Begriff der Recoveryeinheit mittels einer Relation E_α, die im folgenden als *(Recovery-)Einheitsrelation* bezeichnet wird, beschreiben. E_α bildet eine Äquivalenzrelation auf der Menge $W(\alpha)$ der Aktionen einer jeden aktiven Berechnungsfolge α, d.h. sie unterteilt diese Menge in sich nicht überschneidende Klassen.

Definition 3.1.4/1: Sei $P=(A,D,w)$ ein Prozeßsystem, α ein aktiver Ablauf von P. Zwei Aktionen $\alpha[i]$ und $\alpha[j]$ stehen in der *Einheitsrelation* $E_\alpha\subseteq W(\alpha)\times W(\alpha)$ zueinander (Kurzschreibweise: $\alpha[i]E_\alpha\alpha[j]$) genau dann, wenn $\alpha[i]$ und $\alpha[j]$ eine Recoveryeinheit bilden.

Hilfssatz 3.1.4/1: Die Einheitsrelation E_α zu einem aktiven Ablauf α ist eine reflexive, transitive und symmetrische Relation, d.h. eine Äquivalenzrelation.

Beweis: Diese Eigenschaften der Einheitsrelation E_α folgen unmittelbar mit den Definitionen 3.1.3/1 und 3.1.4/1 sowie der Reflexivitäts-, Transitivitäts- und Symmetrie-Eigenschaft der Relationen $proc_P$ und $=$. ■

Beispiel 3.1.4/1: Für zwei Aktionen $a_{i,j}$ und $a_{i,k}$, $i\in\{1,2\}$, $j,k\in\{1,2,3,4\}$, des in Bild 2.2.3.2/1 skizzierten aktiven Ablaufs α des Beispiel-Prozeßsystems P (siehe Bild 2.2.2.4/1) gilt:

$$a_{i,j}E_\alpha a_{i,k} \;\textit{und}\; a_{i,k}E_\alpha a_{i,j}.$$

Die in Abschnitt 2.1.3.1 eingeführte basiert auf Relation B_a und die Einheitsrelation E_a reichen in Kombination aus, um (bei Verwendung der Wiederanlaufalgorithmen 2 2 3 2/1 und 2 2 4 2/1) die Menge der im Fehlerfall zurückzusetzenden Aktionen (formal) vollständig zu beschreiben. Dies wird im folgenden Abschnitt 3.2 gezeigt.

3.2. Beschreibung der Menge zurückzusetzender Aktionen

Um die Menge der im Fehlerfall zurückzusetzenden Aktionen einer Berechnung präzise beschreiben zu können, ist zunächst eine genaue Charakterisierung der im Fehlerfall als fehlerhaft zu bezeichnenden Aktionen, der potentiellen Fehlerverursacher und der gegebenenfalls bereits als korrekt verifizierten Aktionen erforderlich

3.2.1. Fehlerhafte und ungesicherte Aktionen

Als fehlerhaft werden im weiteren Aktionen bezeichnet, die der folgenden Definition genügen.

Definition 3.2.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge von P. Eine Aktion $\alpha[i]$ der Folge α wird als ***fehlerhaft*** bezeichnet genau dann, wenn:

(1) während der Ausführung von $\alpha[i]$ ein Fehler (Prozeß- oder Rechensystemfehler) auftritt, oder

(2) $\alpha[i]$ einem (zugehörigen) Akzeptanztest nicht genügt (genauer: die durch $\alpha[i]$ Variablen zugewiesenen Werte als fehlerhaft erkannt werden).

Kurzschreibweise: $f_\alpha(\alpha[i])=$**true** ::<=> $\alpha[i]$ eine ***fehlerhafte Aktion*** ist.

Für den Fehlerfall ist auch die Unterscheidung in ***gesicherte*** und ***ungesicherte Aktionen*** von Bedeutung, wobei jede Aktion, die der folgenden Definition genügt als ungesichert, andernfalls als gesichert anzusehen ist.

Definition 3.2.1/2: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge von P. Eine Aktion $\alpha[i]$ der Folge α wird als ***ungesichert*** bezeichnet genau dann, wenn:

$$\neg\exists\alpha[j]\in W(\alpha): [proc_P(\alpha[i],\alpha[j]) \;\&\; t(\alpha[j])\geq t(\alpha[i]) \;\&\; up(\alpha[j])=\textbf{true}].$$

Kurzschreibweise: $u_\alpha(\alpha[i])=$**true** ::<=> $\alpha[i]$ eine ***ungesicherte Aktion*** ist.

Die Definition 3.2.1/2 besagt, daß alle Aktionen eines Prozesses P_i, die seit dem Anlegen des aktuellsten Wiederaufsetzpunktes zu P_i ausgeführt worden sind, als ungesichert gelten. Sie müssen im Fehlerfall bei Verlust des Hauptspeicherinhaltes zurückgesetzt werden.

Zu beachten ist, daß prozeßzustandssichernde Aktionen $\alpha[i]$ nach den Annahmen des Abschnittes 2.2.2.1 (der in $wp(P_{bz(\alpha[i])})$ im Zustand α_i gesicherte Befehlszählerzustand zeigt auf die Nachfolgeraktion von $\alpha[i]$ im Prozeß $P_{bz(\alpha[i])}$) nach erfolgreicher Ausführung als gesichert zu betrachten sind.

Beispiel 3.2.1/1: Als fehlerhaft ist im Beispiel 2.2.4.3/1 die Aktion $a_{2,2}$ angenommen, die den im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ als fehlerhaft angesehenen Wert von gv_2 zuletzt modifiziert hat.
Da sowohl für $a_{1,4}$ als auch für $a_{2,4}$ gilt: $wp(a_{1,4})$=**true** bzw. $wp(a_{2,4})$=**true**, sind im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ alle Aktionen gesichert.

Während der Ausführung einer Aktion $\alpha[i]$ auftretende Rechensystemfehler werden nach den Annahmen des Abschnittes 2.2.1.4 immer bereits im Entstehen erkannt, so daß entsprechende Fehlerbehandlungsmaßnahmen unmittelbar nach Auftreten eingeleitet werden können, bevor sich ein Fehler fortpflanzen kann.
Demgegenüber hängt der Zeitpunkt, wann eine Abweichung der Variablenwertänderungen einer Aktion $\alpha[i]$ von spezifizierten Zuständen erkannt wird und ob dies überhaupt geschieht, von der Wirksamkeit von Akzeptanztests ab, die in der Folge α ausgeführt werden. Daher kann durchaus der Fall eintreten, daß für ein Anfangsstück α_{i+j} des Ablaufs α gilt $f_{\alpha_{i+j}}(\alpha[i])$=**false**, obwohl $f_\alpha(\alpha[i])$=**true** ist. Dies bedeutet, daß sich derartige, in der Implementierung eines Prozeßsystems begründete Fehler (Programmierfehler), prinzipiell von einer Aktion zur anderen unerkannt fortpflanzen können. Beim Erkennen von Fehlern dieser Art ist es deshalb wichtig auch ihre potentiellen Verursacher zu lokalisieren.

3.2.2. Potentielle Fehlerverursacher

Bevor die *Menge der potentiellen Verursacher eines Fehlers* jedoch in Definition 3.2.2/4 genau charakterisiert werden kann, sind drei grundlegende Begriffe aus der Relationenalgebra einzuführen: die *Komposition* und die *Vereinigung von Relationen* sowie die Bezeichnung *transitive Hülle einer Relation*.
Die transitive Hülle der elementaren *basiert auf* Relation B_α ist es, die eine Beschreibung der Menge der potentiellen Verursacher eines Fehlers einer Aktion $\alpha[i]$ einer Berechnungsfolge α ermöglicht.

Definition 3.2.2/1: Seien R_1 und R_2 Relationen. Die *Komposition* $R_1{}^\circ R_2$ von R_1 und R_2 ist wie folgt definiert:

$$aR_1{}^\circ R_2b \quad ::<=> \quad \exists c\ (aR_1c\ \&\ cR_2b).$$

Definition 3.2.2/2: Seien R_1 und R_2 zwei Relationen. Dann bezeichnet $R_1 \cup R_2$ die Relation, die durch *Vereinigung* von R_1 und R_2 entsteht, d.h.:

$$aR_1 \cup R_2 b \ ::<=> \ aR_1 b \ \| \ aR_2 b$$

Definition 3.2.2/3: Sei R eine Relation. Dann bezeichnet R^* die *transitive Hülle* von R. Die transitive Hülle R^* von R ist wie folgt definiert:

$$R^* \ = \ R \cup R^2 \cup R^3 \cup R^4 \cup \ \ \cup R^n \cup$$

Mit R^i wird die *i-fache Komposition* von R bezeichnet (z.B. $R^3 ::= R \circ R \circ R$).

Definition 3.2.2/4: Sei α ein aktiver Ablauf von $P=(A,D,w)$. Dann ist die Menge $F_\alpha(\alpha[i])$ der Aktionen, die eine fehlerhafte Ausführung der Aktion $\alpha[i]$ potentiell verursacht haben könnten, wie folgt definiert:

$$F_\alpha(\alpha[i]) ::= \{\alpha[j] \mid \alpha[j] \in W(\alpha) \ \& \ \alpha[i] B_\alpha^* \alpha[j]\} \cup \{\alpha[i]\}.$$

$F_\alpha(\alpha[i])$ wird als die *Menge der potentiellen Verursacher eines Fehlers* der Aktion $\alpha[i]$ bezogen auf die aktive Folge α bezeichnet.

Beispiel 3.2.2/1: Die Zahl der potentiellen Verursacher des Fehlverhaltens der Aktion $a_{2,2}$ in Beispiel 3.2.1/1 ist entsprechend Definition 3.2.2/4:

$$F_\alpha(a_{2,2}) \ = \ \{a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}\}.$$

Dies entspricht der Tatsache, daß in der Situation des Bildes 2.2.4.1/1 im Falle der Berücksichtigung potentieller Fehlerverursacher bei Erkennung des Wertes gw_2 im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ als fehlerhaft sowohl Prozeß P_1 als auch Prozeß P_2 in den Anfangszustand zurückgesetzt werden müssen.

3.2.3. Verifizierte Aktionen

Die im vorangehenden Abschnitt eingeführte Definition zur Beschreibung der Menge der potentiellen Verursacher eines Fehlers umfaßt alle prinzipiell zu verdächtigenden Aktionen. Bei Berücksichtigung potentieller Fehlerverursacher im Rahmen einer Fehlerbehandlung sind jedoch die bereits als korrekt verifizierten Aktionen aus der Menge der verdächtigen Aktionen auszuschließen.

Definition 3.2.3/1: Sei $P=(A,D,w)$ ein Prozeßsystem, $\alpha[i] \subset A$ eine Aktion von P. Sei $V_{\alpha[i]}$ die Menge aller Variablen auf die $\alpha[i]$ zugreift. Dann wird mit $V_{v(\alpha[i])}$ der Anteil von Variablen aus $V_{\alpha[i]}$ bezeichnet, der als *verifiziert* gelten kann und mit $V_{nv(\alpha[i])}$ der Teil

$V_{a[i]} - V_{v(a[i])}$

Entsprechend werden auch die Teilmengen $LV_{v(a[i])}$/ $LV_{nv(a[i])}$ und $SV_{v(a[i])}$/ $SV_{nv(a[i])}$ der verifizierten/nichtverifizierten Variablen von $V_{a[i]}$ bezeichnet, auf die $\alpha[i]$ lesend bzw. schreibend zugreift. Konkret:

$$V_{v(a[i])} ::= (SV_{v(a[i])} \cup LV_{v(a[i])}) - (SV_{nv(a[i])} \cup LV_{nv(a[i])})$$

$$SV_{v(a[i])} ::= \{v \mid \exists j \in \{1,\ldots,|\alpha|\}: [at(\alpha[j])=\textbf{true} \;\&\; v \in LV_{a[j]} \;\&\; \neg\exists k: [v \in SV_{a[k]} \;\&\; v \in SV_{a[i]} \;\&\; j>k>i]]\},$$

$$LV_{v(a[i])} ::= \{v \mid \exists j \in \{1,\ldots,|\alpha|\}: [at(\alpha[j])=\textbf{true} \;\&\; v \in LV_{a[j]} \;\&\; \neg\exists k: [v \in SV_{a[k]} \;\&\; v \in LV_{a[i]} \;\&\; (j>k>i \;||\; i>k>j)]]\},$$

$$SV_{nv(a[i])} ::= SV_{a[i]} - SV_{v(a[i])},$$

$$LV_{nv(a[i])} ::= LV_{a[i]} - LV_{v(a[i])}.$$

Definition 3.2.3/2: Eine Aktion $\alpha[j] \in A$ eines Prozeßsystems $P=(A,D,w)$ basiert in einer aktiven Berechnungsfolge α von P auf der *aus ihrer Sicht verifizierten* Aktion $\alpha[i] \in A$ genau dann, wenn:

$$\alpha[j] B_a \alpha[i] \;\&\; \forall v \in LV_{a[j]} \cap SV_{a[i]}: [\neg\exists \alpha[k] \in W(\alpha): [j>k>i \;\&\; v \in SV_{a[k]}] \Rightarrow v \in LV_{v(a[j])}].$$

Kurzschreibweise $v_a(\alpha[j] B_a \alpha[i])=\textbf{true} :<=> \alpha[j]$ basiert auf der *aus ihrer Sicht verifizierten* Aktion $\alpha[i]$.

Definition 3.2.3/3: Entsprechend der Definitionen 3.2.3/1 und 2 ist die *Menge der nichtverifizierten potentiellen Verursacher* eines Fehlers einer Aktion $\alpha[i]$ innerhalb einer aktiven Berechnung α eines Prozeßsystems $P=(A,D,w)$ in vereinfachter Schreibweise rekursiv definiert als:

$$F_{nv(a)}(\alpha[i]) := \{\alpha[j] \mid \neg v(\alpha[i] B_a \alpha[j]) \;||\; \exists \alpha[k] \in F_{nv(a)}(\alpha[i]). [\neg v(\alpha[k] B_a \alpha[j])]\} \cup \{\alpha[i]\}.$$

Mit $F_{v(a)}(\alpha[i])$ wird im folgenden die Menge der *verifizierten potentiellen Fehlerverursacher* bezeichnet, also die Teilmenge der Aktionen der Menge $F_a(\alpha[i])$, welche nicht zu $F_{nv(a)}(\alpha[i])$ gehören.

Hilfssatz 3.2.3/1: Sei α eine aktive Berechnung eines Prozeßsystems $P=(A,D,w)$, $\alpha[i]$ eine Aktion von α. Dann gilt:

(1) $F_{nv(a)}(\alpha[i]) \subseteq F_a(\alpha[i])$,

(2) $F_{v(a)}(\alpha[i]) = F_a(\alpha[i]) - F_{nv(a)}(\alpha[i])$.

Beweis: Die Beziehungen (1) und (2) folgen unmittelbar aus den Definitionen 3.2.2/4 und 3.2.3/1 bis 3. ■

Beispiel 3.2.3/1: Als verifiziert aus der Sicht der Aktionen $\alpha_{2,j}$, $j \in \{1,2,3,4\}$, gelten im Falle der Berücksichtigung potentieller Fehlerverursacher in der Situation des Bildes 2.2.4.1/2 die Aktionen $\alpha_{1,1}, \alpha_{1,2}, \alpha_{1,3}$ und $\alpha_{1,4}$.

3.2.4. Zurückzusetzende Aktionen

Unter Bezugnahme auf die in den Abschnitten 3.2.1 bis 3.2.3 erfolgten Begriffspräzisierungen, läßt sich schließlich die *Menge der im Falle eines Fehlers zurückzusetzenden Aktionen* einer Berechnungsfolge α sowohl unter dem Aspekt der Nichtberücksichtigung als auch unter dem Gesichtspunkt der Berücksichtigung der (nichtverifizierten) potentiellen Fehlerverursacher, wie Definition und Hilfssatz 3.2.4/2 zeigen werden, adäquat formal beschreiben. Bevor dies jedoch geschieht, soll mittels eines informellen Beweises zu Hilfssatz 3.2.4/1 angedeutet werden, daß sich bei entsprechender Formalisierung des in Abschnitt 2.2 vorgestellten Fehlerbehandlungs-Modells die Menge der im Falle eines Fehlers zurückzusetzenden Aktionen formal begründbar ableiten ließe und deshalb prinzipiell gar nicht explizit definiert werden müßte.
Ein Ansatz zur Formalisierung des Fehlerbehandlungs-Modells wird in Abschnitt 3.4.1 grob skizziert.

Definition 3.2.4/1: Sei α eine aktive Berechnungsfolge. Eine Aktion $\alpha[i] \in W(\alpha)$ muß genau dann zum Wiederaufsetzpunkt $WP_\alpha(\alpha[i])$ oder einen davorliegenden Wiederaufsetzpunkt des Prozesses $P_{bz(\alpha[i])}$ zurückgesetzt werden, wenn gilt: $z_\alpha(\alpha[i])$=**true**.

Hilfssatz 3.2.4/1: Im allgemeinen Fall (d.h. für ein beliebiges Prozeßsystem $P=(A,D,w)$, eine beliebige Berechnung α) gilt für eine beliebige Aktion $\alpha[i]$ einer aktiven Berechnung α bei Nichtberücksichtigung potentieller Fehlerverursacher:

$$z_\alpha(\alpha[i])=\textbf{true} \iff f_\alpha(\alpha[i]) \parallel \exists \alpha[j] \subset W(\alpha): [z_\alpha(\alpha[j]) \;\&\; (\alpha[i] B_\alpha \alpha[j] \parallel \alpha[i] E_\alpha \alpha[j])]$$

Beweis: "<=": Drei Fälle sind zu unterscheiden:

Fall 1 ($f_\alpha(\alpha[i]) \Rightarrow z_\alpha(\alpha[i])$): Angenommen während der Ausführung von $\alpha[i]$ führt ein Fehler (z.B. ein Rechensystemausfall) zum Verlust aller ungesicherten Variablenwerte. Dann kann, unter den gemachten Annahmen zum Fehlerbehandlungs-Modell, der Prozeß $P_{bz(\alpha[i])}$ bestenfalls vom Wiederaufsetzpunkt $WP_\alpha(\alpha[i])$ oder einem davorliegenden Wiederaufsetzpunkt aus fortgesetzt werden. Das heißt es gilt $z_\alpha(\alpha[i])$=**true**.
Im Falle der Fehlerhaftigkeit von $\alpha[i]$ versteht sich die Beziehung $z_\alpha(\alpha[i])$=**true** von

selbst

Fall 2 $((\alpha[i]E_a\alpha[j]\ \&\ z_a(\alpha[j])) \Rightarrow z_a(\alpha[i]))$: Folgt unmittelbar mit der Definition der Relation E_a, da in diesem Falle $proc_P(\alpha[i],\alpha[j])\ \&\ WP_a(\alpha[i])=WP_a(\alpha[j])$ gilt.

Fall 3 $((\alpha[i]B_a\alpha[j]\ \&\ z_a(\alpha[j])) \Rightarrow z_a(\alpha[i]))$: Sei α eine aktive Berechnungsfolge. Bzgl. der beiden Aktionen $\alpha[i],\alpha[j]\in W(\alpha)$ sei gefordert, daß sie die Bedingung: $\alpha[j]$ ***kommt in jeder aktiven Berechnungsfolge vor*** $\alpha[i]$ ***zur Ausführung*** (kurz: $t(\alpha[j])<t(\alpha[i])$) erfüllen

Denkt man sich den Aktionen $\alpha[i]$ und $\alpha[j]$ jeweils eine *private Variable* $v_{\alpha[i]}$ bzw. $v_{\alpha[j]}$ zugeordnet, der durch $\alpha[i]$ bzw. $\alpha[j]$ der Wert "$\alpha[i]$" bzw. "$\alpha[j]$" zugewiesen wird, so könnte die obige Forderung formal durch die folgende Bedingung bzgl. eines (zulässigen) Prozeßsystem-Zustandes $val_P(\alpha)$ repräsentiert werden:

$$(*)\ (v_{\alpha[i]}[val_P(\alpha)]="\alpha[i]") \Rightarrow (v_{\alpha[j]}[val_P(\alpha)]="\alpha[j]").$$

$\alpha[i]$ sei eine Aktion, die sämtlichen globalen und prozeßlokalen Variablen (außer $v_{\alpha[i]}$ und $v_{\alpha[j]}$) den Wert 0 zuweist. Wird nun $\alpha[j]$ zurückgesetzt, muß aufgrund obiger Forderung (Bedingung (*)) auch $\alpha[i]$ zurückgesetzt werden. Da $\alpha[i]$ sämtliche prozeßlokalen und globalen Variablen auf 0 zurückgesetzt hat, kann (unter den gemachten Annahmen zum Fehlerbehandlungs-Modell) ein syntaktisch/berechnungstechnisch konsistenter Prozeßsystem-Zustand nur durch Zurücksetzen des Prozesses $P_{bz(\alpha[i])}$ zum Wiederaufsetzpunkt $WP_a(\alpha[i])$ oder einen davorliegenden Wiederaufsetzpunkt erreicht werden. Das bedeutet $z_a(\alpha[i])=$**true** (da Werte lokaler Variablen nur vor Beginn einer neuen Recoveryeinheit eines Prozesses gesichert werden und nicht vor jeder Aktion, ist in dem eben beispielhaft geschilderten Fall nur *Backward Error Recovery* in bezug auf $\alpha[i]$ möglich, aber kein *Forward Error Recovery*, denn es existiert keine *inverse Funktion* zu $\alpha[i]$, mit der man Änderungen von $\alpha[i]$ rückgängig machen könnte, zur Erläuterung der Begriffe *Backward* und *Forward Error Recovery* siehe Abschnitt 3.4.4).

"$\Rightarrow$": $(\forall\,\alpha[j]\in W(\alpha)\colon [\neg z_a(\alpha[j])\ \|\ (\neg\alpha[i]B_a\alpha[j]\ \&\ \neg\alpha[i]E_a\alpha[j])]\ \&\ \neg f_a(\alpha[i]) \Rightarrow \neg z_a(\alpha[i]))$. Gilt $\neg f_a(\alpha[i])$, so ist das Zurücksetzen von Prozeß $P_{bz(\alpha[i])}$ zum Wiederaufsetzpunkt $WP_a(\alpha[i])$ oder einen davorliegenden Wiederaufsetzpunkt (und damit das Zurücksetzen von $\alpha[i]$) nur dann nicht zu vermeiden, falls es eine Aktion $\alpha[j]$ gibt, die zurückgesetzt werden muß, wobei gilt:

$$[WP_a(\alpha[i])=WP_a(\alpha[j])\ \|\ \exists\alpha[j]\in W(\alpha)\colon [t(\alpha[j])<t(\alpha[i])\ \&\ proc_P(\alpha[i],\alpha[j])]\ \|\ [t(\alpha[j])<t(\alpha[i])\ \&\ \exists v\in SV_{\alpha[j]}\cap LV_{\alpha[i]}\colon [\neg\exists\alpha[k]\in W(\alpha)\colon [t(\alpha[j])<t(\alpha[k])<t(\alpha[i])\ \&\ v\in SV_{\alpha[k]}]]]$$

Drei Fälle sind zu unterscheiden:

Fall 1 $(WP_a(\alpha[i])=WP_a(\alpha[j])\ \&\ z_a(\alpha[j]))$: Aufgrund des zugrundeliegenden Fehlerbehandlungs-Modells kann von $WP_a(\alpha[i])=WP_a(\alpha[j])$ auf $proc_P(\alpha[i],\alpha[j])$ und damit nach Definition 3.1.4/1 auf $\alpha[i]E_a\alpha[j]$ geschlossen werden im Widerspruch zur Annahme:

$$\forall\,\alpha[j]\in W(\alpha)\colon [\neg z_a(\alpha[j])\ \|\ \neg\alpha[i]E_a\alpha[j]].$$

Fall 2 ($\exists\alpha[j]\in W(\alpha)$: $[t(\alpha[j])<t(\alpha[i])$ & $proc_p(\alpha[i],\alpha[j])]$ & $z_a(\alpha[j])$): Da zwei aufeinanderfolgende Aktionen $(\alpha|P_{bz(\alpha[j])})[l]$, $(\alpha|P_{bz(\alpha[j])})[l+1]$ eines Prozesses $P_{bz(\alpha[k])}$ in der Beziehung $(\alpha|P_{bz(\alpha[j])})[l+1]B_a(\alpha|P_{bz(\alpha[j])})_l[l]$ stehen, folgt mit der Transitivität der Eigenschaft $z_a(\ldots)$:

$$\exists\alpha[k]\in W(\alpha): [\alpha[i]B_a\alpha[k] \;\&\; z_a(\alpha[k])]$$

($\alpha[k]$ ist Vorgänger von $\alpha[i]$ bzgl. $P_{bz(\alpha[i])}$ in α). Dies steht im Widerspruch zur Annahme:

$$\forall\alpha[j]\in W(\alpha).\ [\ \neg z_a(\alpha[j]) \parallel \neg\alpha[i]B_a\alpha[j]].$$

Fall 3 ($t(\alpha[j])<t(\alpha[i])$ & $\exists v\in SV_{\alpha[j]}\cap LV_{\alpha[i]}$: $[\neg\exists\alpha[k]\in W(\alpha).\ [t(\alpha_j)<t(\alpha[k])<t(\alpha[i])$ & $v\in SV_{\alpha[k]}]]$ & $z_a(\alpha[j])$) In diesem Falle gilt nach Definition 2.1.3.1/1 $\alpha[i]B_a\alpha[j]$. Zusammen mit der Voraussetzung $z_a(\alpha[j])$ stellt dies einen Widerspruch zur Annahme

$$\forall\alpha[j]\in W(\alpha): [\neg z_a(\alpha[j]) \parallel \neg\alpha[i]B_a\alpha[j]]$$

dar. ■

Ohne Beweis sei an dieser Stelle festgehalten, daß bei Berücksichtigung (nichtverifizierter) potentieller Fehlerverursacher Hilfssatz 3.2.4/1 seine Gültigkeit beibehält, wenn er in folgender Weise modifiziert wird:

(*) $z_a(\alpha[i])$=**true** <=>
$$f_a(\alpha[i]) \parallel \exists\alpha[j]\in W(\alpha): [(z_a(\alpha[j]) \;\&\; (\alpha[i]B_a\alpha[j] \parallel \alpha[i]E_a\alpha[j])) \parallel (f_a(\alpha[j]) \;\&\; \alpha[i]\in F_{nv(a)}(\alpha[j]))].$$

Bei Verlust des Hauptspeicherinhaltes müßte des weiteren die rechte Seite der Äquivalenzbeziehung (*) entsprechend um den Zusatz "$\parallel u_a(\alpha[i])$" ergänzt werden, wenn nicht für das folgende die sich an Definition 3.2.4/2 anschließende Vereinbarung (vgl. dazu weiter unten die Annahme 3.2.4/1) getroffen werden würde.

Unabhängig von dem bisher in Definition und Hilfssatz 3.2.4/1 gesagten, läßt sich die Menge der zurückzusetzenden Aktionen einer durch Rechensystemfehler unterbrochenen oder fehlerhaften Berechnung wie folgt definieren:

Definition 3.2.4/2: Sei α ein aktiver Ablauf eines Prozeßsystems $P=(A,D,w)$ und $\alpha[i]$ eine Aktion der Folge α. Dann ist die Menge $Z_a(\alpha[i])$ der Aktionen, die beim Zurücksetzen von $\alpha[i]$ (zum Wiederaufsetzpunkt $WP_a(\alpha[i])$) ebenfalls zurückgesetzt werden müssen, im Falle der *Nichtberücksichtigung potentieller Fehlerverursacher* definiert als:

$$Z_\alpha(\alpha[i]) ::= \{\alpha[j] \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[j](B_\alpha \cup E_\alpha)^*\alpha[i]\}$$

und im Falle der *Berücksichtigung der potentiellen Fehlerverursacher* definiert als:

$$Z_\alpha(F_{nv(\alpha)}(\alpha[i])) ::= \{\alpha[j] \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[j](B_\alpha \cup E_\alpha)^*\alpha[k] \;\&\; \alpha[k]\in F_{nv(\alpha)}(\alpha[i])\}.$$

Bei Verlust des Hauptspeicherinhaltes im Fehlerfall vergrößern sich die Mengen $Z_\alpha(\alpha[i])$ und $Z_\alpha(F_{nv(\alpha)}(\alpha[i]))$ um den Anteil

$$\{\alpha[j] \mid \alpha[j]\in W(\alpha) \;\&\; \exists\alpha[k]\in W(\alpha): [u_\alpha(\alpha[k]) \;\&\; \alpha[j](B_\alpha \cup E_\alpha)^*\alpha[k]]\}$$

der ungesicherten und der von diesen abhängigen Aktionen.

Bzgl. des Anteils der ungesicherten Aktionen wird im weiteren die folgende Vereinfachung in der Modellierung vorgenommen:

Annahme 3.2.4/1: Jede Unterbrechung einer Berechnung durch einen prozeßsystemtransparent nicht tolerierbaren Fehler, der den Verlust des Hauptspeicherinhalts zur Folge hat, führt zur Einstufung der aktuellsten ungesicherten Aktion eines jeden Prozesses als fehlerhaft.

Annahme 3.2.4/1 beruht auf der Erkenntnis, daß die von einer als fehlerhaft eingestuften ungesicherten Aktion ausgehende Rücksetzausbreitung einer Rücksetzausbreitung ohne Berücksichtigung potentieller Fehlerverursacher gleichgesetzt werden kann und aus diesem Grunde nicht gesondert betrachtet werden muß.

Einen Zusammenhang zwischen Definition 3.2.4/2 und Definition 3.2.4/1 bzw. der Aussage von Hilfssatz 3.2.4/1 stellt der folgende Hilfssatz her:

Hilfssatz 3.2.4/2: Sei $P=(A,D,w)$ ein Prozeßsystem. Im allgemeinen Fall (d.h. für ein beliebiges Prozeßsystem, eine beliebige Berechnung) gilt für eine beliebige Aktion $\alpha[i]$ einer aktiven Berechnung α von P bei Nichtberücksichtigung potentieller Fehlerverursacher:

$$z_\alpha(\alpha[i])=\textbf{true} <=> \alpha[i]\in \bigcup_{\alpha[j]\in W(\alpha)\,\&\, f_\alpha(\alpha[j])} Z_\alpha(\alpha[j])$$

Beweis: "=>" folgt unmittelbar mit der Definition von $Z_\alpha(\alpha[j])$, Hilfssatz 3.2.4/1 und Hilfssatz 3.2.4/5.

Der Hilfssatz 3.2.4/5 stellt in diesem Zusammenhang sicher, daß, unter der Annahme einer endlichen Aktionenmenge $W(\alpha)$, die Tiefe der rekursiven Definition von $z_\alpha(\alpha[i])$=**true** bzw von $Z_\alpha(\alpha[j])$ kleiner $|W(\alpha)|$ ist. Damit kann die Annahme, die Aussage des Hilfssatzes wäre falsch, nach endlich vielen Schritten auf einen Widerspruch geführt werden.

"<=" folgt unmittelbar mit Hilfssatz 3.2.4/1, der Definition von $Z_\alpha(\alpha[j])$, Hilfssatz 3.2.4/5 und der Annahme $f_\alpha(\alpha[j])$=**true** für alle einbezogenen $Z_\alpha(\alpha[j])$. ■

Ein zur Aussage des Hilfssatzes 3.2.4/2 analoger Sachverhalt läßt sich auch für den Fall der Berücksichtigung (nichtverifizierter) potentieller Fehlerverursacher zeigen, ohne dies an dieser Stelle explizit nachzuweisen.

Beispiel 3.2.4/1: Zurückzusetzen sind in der Situation des Bildes 2.2.4.1/2 sowohl bei Berücksichtigung als auch bei Nichtberücksichtigung potentieller Fehlerverursacher die Aktionen $a_{2,1}, a_{2,2}, a_{2,3}$ und $a_{2,4}$. Nach Hilfssatz 3.2.4/2 gilt demnach für diese Aktionen:

$$z_\alpha(a_{2,1})=z_\alpha(a_{2,2})=z_\alpha(a_{2,3})=z_\alpha(a_{2,4})=\textbf{true},$$

wobei $\alpha ::= a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4}$.

Hilfssatz 3.2.4/3: Die Menge der bei Berücksichtigung der (nichtverifizierten) potentiellen Verursacher eines Fehlers einer Aktion $\alpha[i]$ innerhalb einer Berechnung α zurückzusetzenden Aktionen läßt sich auch mittels der folgenden Gleichung charakterisieren:

$$Z_\alpha(F_{nv(\alpha)}(\alpha[i])) = \bigcup_{\alpha[j]\in F_{nv(\alpha)}(\alpha[i])} Z_\alpha(\alpha[j]).$$

Beweis: Folgt unmittelbar mit Definition 3.2.4/2. ■

Für den Beweis des Hilfssatzes 3.2.4/2 war von Bedeutung, daß die Relation $B_\alpha \cup E_\alpha$ eine reflexive Relation ist (Erläuterung: Eine Relation R auf einer Grundmenge A wird als reflexiv bezeichnet genau dann, wenn für ein beliebiges Element $a \in A$ gilt: aRa). Denn für auf einer endlichen Grundmenge A definierte reflexive Relationen R läßt sich ein Index i finden, so daß gilt: $R^* = R^i$. Dieser Sachverhalt wird in Hilfssatz 3.2.4/5 bewiesen. Zunächst aber wird die Reflexivität der Relation $B_\alpha \cup E_\alpha$ gezeigt.

Hilfssatz 3.2.4/4: Sei α eine aktive Berechnung eines Prozeßsystems $P=(A,D,w)$ Die

Relation $(E_\alpha \cup B_\alpha)^*$ ist eine reflexive Relation.

Beweis: Folgt unmittelbar aufgrund der Reflexivität der Relation E_α, denn mit $\alpha[i] E_\alpha \alpha[i]$ folgt

$$(\alpha[i] E_\alpha \alpha[i] \;||\; \alpha[i] B_\alpha \alpha[i]) = \textbf{true}$$

und damit nach Definition 3.2.2/2 $\alpha[i](E_\alpha \cup B_\alpha)\alpha[i]$. Da nach Definition 3.2.2/3 gilt $(E_\alpha \cup B_\alpha) \subseteq (E_\alpha \cup B_\alpha)^*$, folgt damit die Reflexivität von $(E_\alpha \cup B_\alpha)^*$. ■

Hilfssatz 3.2.4/5: Sei R reflexive Relation auf $A \times A$ und $|A| = n > 1$; der Fall $n = 1$ ist ohne Bedeutung. Dann existiert ein i, $0 < i < n$, so daß gilt $R^* = R^i$.

Beweis: Da aufgrund der angenommenen Reflexivität von R trivialerweise $R^{i+1} \supseteq R^i$ für $i > 0$ gilt, genügt es zu zeigen, daß für alle $i > n-1$ die Beziehung $R^{n-1} = R^i$ erfüllt ist. Letzteres folgt unmittelbar mit dem Nachweis der Gleichheit $R^{n-1} = R^n$, die, wie nachstehend skizziert, bewiesen wird:

$R^{n-1} \subseteq R^n$: Folgt aufgrund der Reflexivität von R.

$R^{n-1} \supseteq R^n$ Angenommen es gelte $a R^n b$ Dann existieren nach Definition einer n-fachen Komposition Elemente $c_1, c_2, \ldots, c_{n-1} \in A$, so daß gilt:

$$a = c_0 R c_1,\ c_1 R c_2,\ \ldots,\ c_{n-1} R c_n = b.$$

Da $a = c_0, c_1, c_2, c_3, \ldots, c_{n-1}, c_n = b$ insgesamt $n+1$ Elemente sind, müssen wegen der Annahme $|A| = n$ mindestens zwei von ihnen identisch sein. O.B.d.A. seien dies c_j und c_{j+k}, $k > 0$, $j \geq 0$ & $j + k \leq n$
Mit $c_j = c_{j+k}$ folgt jedoch die Gültigkeit der Beziehung $a R^{n-k} b$ und somit wegen $R^{n-1} \supseteq R^{n-k}$ (Reflexivität von R) auch $a R^{n-1} b$, und der Hilfssatz ist damit bewiesen. ■

Auf die Aussage von Hilfssatz 3.2.4/5 wird auch noch in späteren Kapiteln zurückgegriffen, denn mittels dieses Hilfssatzes läßt sich der folgende Sachverhalt zeigen:

Hilfssatz 3.2.4/6: Für jede endliche aktive Berechnungsfolge α eines Prozeßsystems $P = (A, D, w)$ gilt:

$$\forall \alpha[i], \alpha[j] \in W(\alpha): [(\alpha[j](B_\alpha \cup E_\alpha)^* \alpha[i]) \Rightarrow \exists \alpha[j_1], \ldots, \alpha[j_m] \in W(\alpha): [\alpha[j] = \alpha[j_m](B_\alpha \cup E_\alpha)\alpha[j_{m-1}](B_\alpha \cup E_\alpha) \ldots \alpha[j_2](B_\alpha \cup E_\alpha)\alpha[j_1] = \alpha[i]]].$$

Beweis: Folgt unmittelbar mit Hilfssatz 3.2.4/5, denn, da E_α eine reflexive Relation ist, ist auch die Relation $(B_\alpha \cup E_\alpha)^* \subseteq W(\alpha)^2$ eine reflexive Relation, und $W(\alpha)$ ist nach Voraus-

setzung endlich. Nach Hilfssatz 3.2.4/5 muß also ein $m \in \mathbb{N}$ existieren, so daß gilt $\alpha[i](B_\alpha \cup E_\alpha)^m \alpha[j]$. Die Aussage des Satzes folgt schließlich mit der Definition der m-fachen Komposition $(B_\alpha \cup E_\alpha)^m$ von $(B_\alpha \cup E_\alpha)$ (siehe Definition 3.2.2/1). ■

3.3. Klassifizierung von Prozeßsystem-Zuständen

Analog zur Einteilung von Aktionen einer Berechnung in (potentiell) fehlerhafte und deshalb zurückzusetzende sowie verifizierte Aktionen, lassen sich, neben berechnungstechnisch/syntaktisch konsistenten, auch fehlerhafte und aus diesem Grunde zu korrigierende sowie verifizierte Prozeßsystem-Zustände unterscheiden. Auf die Klassifizierung von Prozeßsystem-Zuständen geht dieser Abschnitt näher ein.

3.3.1. Konsistente Prozeßsystem-Zustände

Als *berechnungstechnisch/syntaktisch konsistent* wird ein Prozeßsystem-Zustand $val_P(\alpha)$ genau dann bezeichnet, wenn er der Definition 3.3.1/1 genügt.

Definition 3.3.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Der Zustand $val_P(\alpha)$ wird genau dann als (*berechnungstechnisch/syntaktisch*) *konsistent* bezeichnet, wenn α ein aktiver Ablauf in P ist

Beispiel 3.3.1/1: Der im Bild 2.2.2.4/10 dargestellte Prozeßsystem-Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ ist nach Definition 3.3.1/1 als konsistent zu bezeichnen, da er den Zustand nach Abarbeitung einer aktiven Berechnung des Beispiel-Prozeßsystems P (vgl. Bild 2.2.2.4/1) repräsentiert.

3.3.2. Fehlerhafte und ungesicherte Prozeßsystem-Zustände

Der Begriff fehlerhafter Prozeßsystem-Zustand, eine erste Grobdefinition wurde in Abschnitt 2.2.3.1 gegeben, läßt sich vor dem Hintergrund der Ausführungen des Abschnittes 3.2 wie folgt präzisieren:

Definition 3.3.2/1: Sei α eine aktive Berechnung eines Prozeßsystems $P=(A,D,w)$, $val_P(\alpha)$ der zugehörige Prozeßsystem-Zustand nach Ablauf von α. Dann wird $val_P(\alpha)$ als *fehlerhaft* bezeichnet genau dann, wenn gilt:

$$\exists \alpha[i] \in W(\alpha),\ i \in \{1,\ldots,|\alpha|\}.\ \ f_\alpha(\alpha[i])=\textbf{true}.$$

Kurzschreibweise: $f_\alpha(val_P(\alpha))=\textbf{true}$ ::<=> $val_P(\alpha)$ ist fehlerhaft.

Definition 3.3.2/2: Sei α eine aktive Berechnung eines Prozeßsystems $P=(A,D,w)$. $val_P(\alpha)$ sei der zugehörige Prozeßsystem-Zustand nach Ablauf von α. Dann wird

$val_P(\alpha)$ als *ungesichert* bezeichnet genau dann, wenn gilt:

$$\exists a[i] \in W(\alpha),\ i \in \{1,\ldots,|\alpha|\}: u_a(a[i])=\textbf{true}.$$

Kurzschreibweise: $u_a(val_P(\alpha))=\textbf{true}$ ::<=> $val_P(\alpha)$ ist ungesichert.

Definition 3.3.2/3: Ist $val_P(\alpha)$ ein fehlerhafter Prozeßsystem-Zustand, und ist der Wert der Variablen v im Zustand $val_P(\alpha)$ fehlerhaft, so wird dies im folgenden durch die Schreibweise $f_a(v[val_P(\alpha)])=\textbf{true}$ zum Ausdruck gebracht.

Beispiel 3.3.2/1: Wird die Variable gv_2 im Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ als fehlerhaft erkannt, so gilt dieser im Sinne von Definition 3.3.2/1 als fehlerhaft, obwohl er im Sinne von Definition 3.3.1/1 als berechnungstechnisch/syntaktisch konsistent bezeichnet werden kann (vgl. hierzu Beispiel 2.2.3.3/2).

3.3.3. Verifizierte Prozeßsystem-Zustände

Als (im weiteren Sinne) semantisch korrekt können Prozeßsystem-Zustände $val_P(\alpha)$ bezeichnet werden, die durch Akzeptanztests verifiziert wurden und deshalb aus Sicht der an der Berechnung α beteiligten Aktionen als fehlerfrei gelten können

Definition 3.3.3/1: Sei α eine beliebige aktive Aktionenfolge eines Prozeßsystems $P=(A,D,w)$. Der Zustand $val_P(\alpha)$ gilt aus der Sicht der Aktionen des Ablaufs α als verifiziert genau dann, wenn:

$$\forall a[i],a[j],\ i,j \subset \{1,\ldots,|\alpha|\}:\ v(a[i]B_\alpha a[j])=\textbf{true}.$$

Kurzschreibweise: $v_a(val_P(\alpha))=\textbf{true}$ ::<=> $val_P(\alpha)$ ist verifiziert.

Beispiel 3.3.3/1: Der zu dem im Bild 2.2.4.1/2 dargestellten Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ rekonstruierte Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4})$ kann im Sinne der Definition 3.3.3/1 genaugenommen nicht als Beispiel eines verifizierten Zustandes gelten, da die durch $a_{1,1}$, $a_{1,2}$ und $a_{1,3}$ vorgenommenen Änderungen des Befehlszählers bz_1 nicht durch Akzeptanztests verifiziert wurden.

Das Beispiel 3.3.3/1 zeigt, daß die Eigenschaft eines Zustandes verifiziert zu sein sehr streng ist. Durch entsprechende Annahmen (z.B. Befehlszählerstände sind generell als korrekt zu betrachten), läßt sie sich aber ohne weiteres geeignet abschwächen. Zu beachten ist jedoch, daß selbst verifizierte Prozeßsystem-Zustände $val_P(\alpha)$ durch Rechensystemfehler "unterbrochen" und zu im Sinne von Definition 3.3.2/1

fehlerhaften Prozeßsystem-Zuständen führen können, da sie in der Regel, gemäß Definition 3.3.2/2 als ungesichert zu bezeichnen sind, und aus diesem Grunde zunächst ein berechnungstechnisch/syntaktisch konsistenter Zustand errechnet werden muß, bevor die unterbrochene Berechnung fortgesetzt werden kann.

3.3.4. Korrektur fehlerhafter Prozeßsystem-Zustände

Die Existenz verifizierter Prozeßsystem-Zustände wird im allgemeinen die Ausnahme sein, denn sie stellt hohe Anforderungen an die Zuverlässigkeit und Wirksamkeit von Akzeptanztests, wie sie in realen Systemen kaum zu erbringen sind. In dieser Arbeit wird aus diesem Grunde nicht davon ausgegangen, daß von jedem gegebenen Prozeßsystem-Zustand feststellbar ist, ob er (auch semantisch, gemäß einer vorliegenden Spezifikation des Systemverhaltens) als korrekt gelten kann oder nicht. Daher kann die Korrektur von als fehlerhaft erkannten Prozeßsystem-Zuständen nur in der Rekonstruktion von berechnungstechnisch/syntaktisch konsistenten Prozeßsystem-Zustanden, die unter Zugrundelegung der Definition 3.3.2/1 als nicht fehlerhaft zu bezeichnen sind, bestehen. Solche Prozeßsystem-Zustände werden im folgenden als *(potentiell) fehlerfrei* bezeichnet.

Definition 3.3.4/1: Sei α ein aktiver Ablauf eines Prozeßsystems $P=(A,D,w)$. Dann gilt der Prozeßsystem-Zustand $val_P(\alpha)$ als *(potentiell) fehlerfrei* genau dann, wenn es keine Aktionen $\alpha[i]\in W(\alpha)$ gibt, für die gilt $z_\alpha(\alpha[i])=$**true**.

Hilfssatz 3.3.4/1: Der Prozeßsystem-Zustand $val_P(\alpha)$ ist im allgemeinen Fall bei Nichtberücksichtigung potentieller Fehlerverursacher (potentiell) fehlerfrei genau dann, wenn:

$$\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} Z_\alpha(\alpha[i]) = \phi.$$

Bei Berucksichtigung (nichtverifizierter) potentieller Fehlerverursacher ist $val_P(\alpha)$ (potentiell) fehlerfrei genau dann, wenn:

$$\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} Z_\alpha(F_{nv(\alpha)}(\alpha[i])) = \phi.$$

Beweis: Folgt unmittelbar mit der Definition eines potentiell fehlerfreien Prozeßsystem-Zustandes (siehe Definition 3.3.4/1) und der Definition von $Z_\alpha(\alpha[i])$ bzw. $Z_\alpha(F_{nv(\alpha)}(\alpha[i]))$ (siehe Definition 3.2.4/2) und Hilfssatz 3.2.4/2. ■

Beispiel 3.3.4/1: Der im Beispiel 2.2.4.3/2 zu dem, gemäß Annahme, fehlerhaften

Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{2,1}a_{2,2}a_{2,3}a_{2,4})$ rekonstruierte Zustand $val_P(a_{1,1}a_{1,2}a_{1,3}a_{1,4})$ ist (potentiell) fehlerfrei.

3.4. Ableitung von Recoveryfunktionen

Auf welche Weise zu einem fehlerhaften Prozeßsystem-Zustand ein berechnungstechnisch/syntaktisch konsistenter und im Sinne von Definition 3.3.2/1 nicht fehlerhafter Prozeßsystem-Zustand rekonstruiert werden kann, wird im folgenden durch Abbildungsvorschriften, die auf Berechnungsfolgen bzw. Prozeßsystem-Zuständen definiert sind und als *Recoveryfunktionen* (*Fehlerbehandlungsfunktionen*) bezeichnet werden, beschrieben.

Zunächst werden Anforderungen an Recoveryfunktionen motiviert und präzisiert. Im zweiten Schritt wird eine Recoveryfunktion, die unabhängig von der Art eines innerhalb einer Berechnung auftretenden Fehlers zu seiner Korrektur eingesetzt werden kann, abgeleitet. Schließlich werden zwei spezielle Recoveryfunktionen zur Beseitigung von Synchronisationsfehlern und zur Behandlung von Kommunikationsfehlern vorgestellt.

3.4.1. Anforderungen an Recoveryfunktionen

Eine *Recoveryfunktion (Fehlerbehandlungsfunktion)* r, mit der ein fehlerhafter Systemzustand $val_P(\alpha)$ in einen (potentiell) fehlerfreien Zustand $val_P(r(\alpha))$ überführt werden kann, sollte die folgenden drei Bedingungen erfüllen:

(1) $r(\alpha)$ muß aktiv sein,

(2) $r(val_P(\alpha)) = val_P(r(\alpha))$ und

(3) $$\bigcup_{a[i] \in \Psi(r(\alpha)) \& f_{r(\alpha)}(a[i])} Z_{r(\alpha)}(a[i]) = \phi.$$

Die beiden Bedingungen (1) und (2) stellen gemeinsam sicher, daß die Fehlerkorrektur zu einem Zustand führt, der vom Anfangszustand $val_P(\lambda)$ über eine aktive Berechnung erreichbar ist.
Einhaltung von Bedingung (2) gewährleistet zusätzlich, daß, unter den gemachten Annahmen des Fehlerbehandlungs-Modells, vom aktuellen (fehlerhaften) Prozeßsystem-Zustand $val_P(\alpha)$ ausgehend, mit Hilfe der in den einzelnen Prozessen eingerichteten Wiederaufsetzpunkte und der Variablenhistorie zu den globalen Variablen, der Prozeßsystem-Zustand $val_P(r(\alpha))$ rekonstruiert werden kann.
Bedingung (3) stellt schließlich sicher, daß der rekonstruierbare Prozeßsystem-Zustand $val_P(r(\alpha))$ (potentiell) fehlerfrei ist.

Diese Überlegungen führen zu der folgenden Definition:

Definition 3.4.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Sei $val_P(\alpha)$ ein fehlerhafter Prozeßsystem-Zustand. Dann ist r eine *Recoveryfunktion* zu einer aktiven Berechnung α, $r ::= r_A \cup r_D$, genau dann, wenn:

(1) $r(\alpha)$ aktiv,

(2) $r(val_P(\alpha))=val_P(r(\alpha))$,

(3) $\bigcup_{\alpha[i]\in \Psi(r(\alpha)) \;\&\; f_{r(\alpha)}(\alpha[i])} Z_{r(\alpha)}(\alpha[i])=\phi$,

wobei r_A und r_D zwei auf Berechnungsfolgen bzw. Prozeßsystem-Zuständen definierte Abbildungen sind:

$$r_A \; A^* \to A^* \quad \text{und} \quad r_D: D \to D,$$

$r(\alpha) ::= r_A(\alpha)$ und $r(val_P(\alpha)) ::= r_D(val_P(\alpha))$.

Im weiteren werden nun verschiedene Recoveryfunktionen r_0, r_1, r_2 und r_3 vorgestellt, die Funktion r_0 im folgenden Beispiel und die Recoveryfunktionen r_1, r_2, r_3 nacheinander in den nächsten drei Abschnitten.

Beispiel 3.4.1/1: Die wie folgt definierte Recoveryfunktion r_0:

$$r_0(\alpha) ::= \lambda \quad \& \quad r_0(val_P(\alpha)) ::= val_P(\lambda)=val_P(r_0(\alpha)),$$

für alle aktiven Berechnungsfolgen α eines Prozeßsystems, erfüllt trivialerweise (unter den gemachten Annahmen des Fehlerbehandlungs-Modells) bei vorausgesetztem berechnungstechnisch/syntaktisch konsistenten Anfangszustand $val_P(\lambda)$ die Bedingungen (1), (2) und (3). Sie kann als *triviale Recoveryfunktion* r_0 bezeichnet werden.

Bemerkung: Um zu einem fehlerhaften Berechnungsablauf α mittels einer Recoveryfunktion r einen syntaktisch/berechnungstechnisch konsistenten Zustand $r(val_P(\alpha))=val_P(r(\alpha))$ rekonstruieren zu können, ist im allgemeinen, wie in Abschnitt 2.2 skizziert, der Rückgriff auf eine zu α zu führende Prozeßsystemzustands-Historie erforderlich. Wird die aktuelle zu einem aktiven Ablauf α eines Prozeßsystems $P=(A,D,w)$ aufgebaute Prozeßsystemzustands-Historie mit $h_P(\alpha)$ bezeichnet, so muß eine Recoveryfunktion r natürlich auch die Forderung:

$$(4)\quad r(h_P(\alpha)) = h_P(r(\alpha))$$

erfüllen. Forderung (4) stellt sicher, daß zu einem mittels r rekonstruierten Prozeßsystem-Zustand $r(val_P(\alpha)) = val_P(r(\alpha))$ auch eine passende Prozeßsystemzustands-Historie $h_P(r(\alpha))$ abgeleitet werden kann, so daß eine Fehlerbehandlung ohne Verlust der Möglichkeit zur Korrektur nachfolgend auftretender Fehler erfolgen kann.
Für die im weiteren vorgestellten Recoveryfunktionen wird auf den Nachweis der Eigenschaft (4) verzichtet, da dies zum einen von der Beweistechnik her keine neuen Aspekte mit sich bringen und zum anderen vom Umfang her den Rahmen dieser Arbeit sprengen würde. An dieser Stelle soll lediglich kurz angedeutet werden, wie eine Formalisierung des Konzeptes Prozeßsystemzustands-Historie, die gegebenenfalls für eine zu untersuchende Recoveryfunktion r einen Beweis von (4) zuläßt, aussehen könnte.
Eine Formalisierung des Konzeptes der Prozeßsystemzustands-Historie kann sich an dem in Abschnitt 2.2 vorgestellten Fehlerbehandlungs-Modell orientieren. In diesem Falle besteht eine Prozeßsystemzustands-Historie H aus, bei n Prozessen, n Listen lokaler Prozeßzustände $L_{h(P_i)}$ und, bei m globalen Variablen, m Listen von Historien $L_{h(gv_j)}$ der jeweiligen Wertänderungen der globalen Variablen, d.h. $H \subseteq (L_{h(P_i)})^n \times (L_{h(gv_j)})^m$. Ein Element einer Liste $L_{h(P_i)}$ ist ein Tripel, dessen Komponenten wiederum Listen sind, nämlich die Liste der lokalen Variablenwerte des Prozesses P_i in einem aufgezeichneten Prozeßsystem-Zustand, die Liste der globalen Lesereferenzen zwischen zwei benachbarten Wiederaufsetzpunkten von P_i und die Liste der globalen Schreibreferenzen zwischen zwei benachbarten Wiederaufsetzpunkten von P_i. Analog läßt sich ein Element $L_{h(gv_j)}$ auch als ein Tripel darstellen, wobei die erste Komponente den gesicherten Variablenwert, Komponente zwei die zugehörige Schreibreferenz zu einem Wiederaufsetzpunkt und die dritte Komponente eine Liste von Lesereferenzen zu Wiederaufsetzpunkten darstellen.
Sei $D_H = (L_{h(P_i)})^n \times (L_{h(gv_j)})^m$. Analog zur Funktion val_P, ordnet die obenerwähnte Funktion h_P Ablaufen α eines Prozeßsystems $P=(A,D,w)$ Zustände aus D_H zu, und der auf Prozeßsystemzustands-Historien definierte Teil r_{D_H} einer Recoveryfunktion r ist eine Abbildung von D_H nach D_H. Eine Recoveryfunktion r läßt sich demnach, bei einer entsprechenden (nahtlosen) Erweiterung des Prozeßsystem-Modells um eine Komponente Zustands-Historie, als Vereinigung dreier Funktionen r_A, r_D und r_{D_H} darstellen.
Soweit zur Formalisierung des Konzeptes Prozeßsystemzustands-Historie. Dem interessierten Leser bleibt es überlassen, detailliertere Überlegungen in dieser Richtung anzustellen.

3.4.2. Allgemeine Recoveryfunktion

Eine nichttriviale Recoveryfunktion, die alle in dieser Arbeit als relevant identifizierten Fehlersituationen (vgl. Abschnitt 2.2.3.1) zu korrigieren gestattet, ist die im Rahmen des Satzes 3.4.2/1 definierte Funktion r_1. Beim Beweis dieses Satzes wird das nachstehend abgeleitete Korollar 3.4.2/2 benutzt, das von der folgenden Definition Gebrauch macht:

Definition 3.4.2/1: Der *unmittelbare Vorgänger* einer Aktion $\alpha[i]$ in einer Berechnung α bezogen auf einen Prozeß P_j wird mit $vg_\alpha(\alpha[i]|P_j)$ bezeichnet. Entsprechend wird der *unmittelbare Nachfolger* von $\alpha[i]$ innerhalb einer Berechnung α bezogen auf einen Prozeß P_j mit $nf_\alpha(\alpha[i]|P_j)$ bezeichnet.

Hilfssatz 3.4.2/1: Sei α eine aktive Berechnungsfolge und $\alpha[i]\in W(\alpha)$. Existiert ein unmittelbarer Vorgänger von $\alpha[i]$ in α bezogen auf den Prozeß $P_{bz(\alpha[i])}$ (kurz: $vg_\alpha(\alpha[i] \mid P_{bz(\alpha[i])})$), so gilt:

$$\alpha[i]B_\alpha vg_\alpha(\alpha[i]|P_{bz(\alpha[i])}).$$

Beweis: Da $\alpha[i]$ und $vg_\alpha(\alpha[i]|P_{bz(\alpha[i])})$ nach Voraussetzung zum gleichen Prozeß gehören, existiert eine Variable v, nämlich der Befehlszähler $bz_{bz(\alpha[i])}$, so daß gilt:

$$(1)\ bz_{bz(\alpha[i])}\in LV_{\alpha[i]}\cap SV_{vg_\alpha(\alpha[i]|P_{bz(\alpha[i])})}\neq\phi$$

Da die Aktion $vg_\alpha(\alpha[i]|P_{bz(\alpha[i])})$ unmittelbarer Vorgänger von $\alpha[i]$ bezogen auf Prozeß $P_{bz(\alpha[i])}$ ist, folgt mit v repräsentiert durch $bz_{bz(\alpha[i])}$:

$$(2)\ \neg\exists\alpha[j]\colon [t(vg_\alpha(\alpha[i]|P_{bz(\alpha[i])}))<t(\alpha[j])<t(\alpha[i])\ \&\ bz_{bz(\alpha[i])}\in SV_{\alpha[j]}].$$

Nach Definition der Relation B_α gilt:

$$\alpha[i]B_\alpha vg_\alpha(\alpha[i]|P_{bz(\alpha[i])}) ::<=>((1)\&(2)).\ \blacksquare$$

Korollar 3.4.2/1: Sei α eine aktive Berechnungsfolge und $\alpha[i]\in W(\alpha)$. Existiert ein unmittelbarer Nachfolger von $\alpha[i]$ in α bezogen auf Prozeß $P_{bz(\alpha[i])}$ (kurz: $nf_\alpha(\alpha[i]|P_{bz(\alpha[i])})$), so gilt:

$$nf_\alpha(\alpha[i]|P_{bz(\alpha[i])})B_\alpha\alpha[i].$$

Beweis: Folgt unmittelbar mit Hilfssatz 3.4.2/1. $\blacksquare$

Korollar 3.4.2/2: Sei α eine aktive Berechnungsfolge und $\alpha[i]\in W(\alpha)$. Existiert ein Vorgänger $\alpha[j]$ (bzw. Nachfolger $\alpha[j]$) von $\alpha[i]$ in α bezogen auf den Prozeß $P_{bz(\alpha[i])}$, so gilt:

$$\alpha[i]B_\alpha^*\alpha[j] \text{ (bzw. } \alpha[j]B_\alpha^*\alpha[i]).$$

Beweis: Die Behauptung folgt unmittelbar mit Hilfssatz 3.4.2/1 (bzw. Korollar 3.4.2/1) und der Transitivität von B_α^*. $\blacksquare$

Satz 3.4.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Eine *nichttriviale Recoveryfunktion* für aktive Berechnungen α von P stellt die wie folgt definierte Funktion r_1 dar:

$r_1(\lambda) ::= \lambda$ und $r_1(\alpha) ::= r_1(\alpha[1])...r_1(\alpha[|\alpha|])$, $\alpha[i] \in W(\alpha)$, $i \in \{1,...,|\alpha|\}$,

und

$$r_1(\alpha[i]) := \begin{cases} \lambda & z_a(\alpha[i]) = \textbf{true} \\ & falls \\ \alpha[i] & sonst \end{cases}$$

Beweis: Es muß gezeigt werden, daß r_1 alle drei Bedingungen der Definition 3.4.1/1 erfüllt, falls α aktiv ist:

(1) $r_1(\alpha)$ *ist aktiv*: Angenommen $r_1(\alpha)$ wäre nicht aktiv. Dann müßte es ein i geben, $1 \le i \le |\alpha|$, so daß gilt: α_i aktiv, $r_1(\alpha_{i-1})$ aktiv und $r_1(\alpha_{i-1})r_1(\alpha[i])$ nicht aktiv. ($r_1(\alpha[i])$ wäre in diesem Fall zwangsläufig ungleich λ, sonst wäre nämlich $r_1(\alpha_{i-1})r_1(\alpha[i])$ aktiv).
Das heißt aber, daß es ein v und eine Aktion $\alpha[j]$, $0 < j < i$, geben müßte, $v \in LV_{a[i]}$, so daß gilt:

$$v[val_P(\alpha_j)] \neq v[val_P(r_1(\alpha_j))] \;\&\; \alpha[i] B_a \alpha[j].$$

$r_1(\alpha[j])$ mußte ungleich λ sein, sonst würde aufgrund der Beziehung $\alpha[i] B_a \alpha[j]$ auch $r_1(\alpha[i]) = \lambda$ gelten im Widerspruch zur Annahme.
Dies wiederum bedeutet, daß ein weiteres v und eine Aktion $\alpha[k]$, $0 \le k < j$, existieren müßten, so daß gilt:

$$v[val_P(\alpha_k)] \neq v[val_P(r_1(\alpha_k))] \;\&\; \alpha[j] B_a \alpha[k].$$

Analog zu der vorangehenden Überlegung kann $r_1(\alpha[k]) \neq \lambda$ nachgewiesen werden. Diese Schlußweise führt spätestens nach $i+1$ Schritten aufgrund der Beziehung $val_P(\lambda) = val_P(r_1(\lambda))$ auf einen Widerspruch.

(2) $r_1(val_P(\alpha)) = val_P(r_1(\alpha))$ *ist vom Zustand* $val_P(\alpha)$ *unter den Annahmen des zugrundegelegten Fehlerbehandlungs-Modells rekonstruierbar*: Im ersten Schritt wird nachgewiesen, daß ausgehend vom Prozeßsystem-Zustand $val_P(\alpha)$ ein Prozeßsystem-Zustand $r_1(val_P(\alpha))$ rekonstruiert werden kann, von dem im zweiten Schritt bewiesen wird, daß er mit $val_P(r_1(\alpha))$ übereinstimmt.

Schritt 1 Sei $V_P := \bigcup_{a \in A} V_a$ die Menge aller Variablen des Prozeßsystems $P = (A, D, w)$.
Für lokale Variablen $v \in V_P$ wird ein - wie später bewiesen - (potentiell) fehlerfreier Zustand wie folgt rekonstruiert: Ist $\alpha[j]$, $j \le |\alpha|$, die älteste Aktion des Prozesses $P_{bz(\alpha[j])}$, d.h. j minimal, für die $r_1(\alpha[j]) = \lambda$ gilt, und, gehört v zu $P_{bz(\alpha[j])}$ ($v \in V_{P_{bz(\alpha[j])}}$), so kann mit Hilfe des Wiederaufsetzpunktes $WP_a(\alpha[j])$ in bezug auf alle lokalen Variablenwerte des Prozesses $P_{bz(\alpha[j])}$ der Prozeßsystem-Zustand $v[val_P(\alpha_{j-1})]$ wiederhergestellt werden.
Für globale Variablen $v \in V_P$ wird ein - wie später bewiesen wird - (potentiell) fehlerfreier Zustand wie folgt rekonstruiert. Sei $\alpha[j]$ die jüngste Aktion, d.h. j maximal (aber $j < |\alpha|$), welche die globale Variable v verändert hat, d.h. $v \in SV_{\alpha[j]}$, und, für die $r_1(\alpha[j]) \neq \lambda$ gilt. Mit Hilfe der Variablenhistorie zu der globalen Variablen v läßt sich ermitteln, welcher Wert v durch $\alpha[j]$ zugewiesen wurde. Das heißt

$v[r_1(val_P(\alpha))]$ kann der Wert $v[val_P(\alpha_j)]$ zugewiesen werden.
Mit Hilfe dieser Erkenntnisse läßt sich $v[r_1(val_P(\alpha))]$ für alle $v \in V_P$ eines Prozeßsystems $P=(A,D,w)$ bzgl. der aktiven Berechnungsfolge α wie folgt definieren:

$$v[r_1(val_P(\alpha))] ::= \begin{cases} v[val_P(\alpha_{i-1})] & l(v) \;\&\; v \in V_{P_{bs(\alpha[i])}} \;\&\; r_1(\alpha[i])=\lambda \;\&\; i \textit{ minimal} \\ v[val_P(\alpha)] & l(v) \;\&\; \neg \exists \alpha[i] \in W(\alpha)[v \in V_{\alpha[i]} \;\&\; r_1(\alpha[i])=\lambda] \\ \textit{falls} & \\ v[val_P(\alpha_i)] & g(v) \;\&\; v \in SV_{\alpha[i]} \;\&\; r_1(\alpha[i]) \neq \lambda \;\&\; i \textit{ maximal} \\ v[val_P(\lambda)] & g(v) \;\&\; \neg \exists \alpha[i] \in W(\alpha)[v \in SV_{\alpha[i]} \;\&\; r_1(\alpha[i]) \neq \lambda] \end{cases}$$

Schritt 2: Zu zeigen bleibt, daß für alle $v \in V_P$ gilt:

$$v[r_1(val_P(\alpha))]=v[val_P(r_1(\alpha))]$$

Angenommen es würde $v[r_1(val_P(\alpha))] \neq v[val_P(r_1(\alpha))]$ für ein $v \in V_P$ gelten. Vier Fälle sind zu unterscheiden:

Fall 1 ($l(v)$=**true** & $v \in V_{P_{bs(\alpha[i])}}$ & $r_1(\alpha[i])=\lambda$ & i *minimal*): Nach Definition von r_1 gilt in diesem Fall:

$$v[r_1(val_P(\alpha))]::=v[val_P(\alpha_{i-1})].$$

Zu zeigen bleibt, daß $v[val_P(r_1(\alpha))]$ gleich $v[val_P(\alpha_{i-1})]$ ist.
Angenommen dies wäre nicht der Fall. Dann muß es ein minimales j geben, $v \in SV_{\alpha[j]}$: $j<i$ ($j \geq i$ würde, da v eine lokale Variable ist, nach Korollar 3.4.2/2 $\alpha[j]B_a\alpha[i]$ $z_a(\alpha,j)$=**true** und somit $r_1(\alpha[j])=\lambda$ bzw. im Falle $i=j$ ebenfalls $r_1(\alpha[j])=\lambda$ bedeuten), für das gilt:

$$v[val_P(r_1(\alpha_j))] \neq v[val_P(\alpha_j)] \;\&\; r_1(\alpha[j]) \neq \lambda.$$

Aufgrund der Minimalität von j kann o B d A angenommen werden, daß für alle lokalen Variablen $v' \in V_{P_{bs(\alpha[j])}}$ gilt:

$$v'[val_P(\alpha_k)]=v'[val_P(r_1(\alpha_k))],\ k<j.$$

Das heißt aber, es muß eine globale Variable $v_{i_j} \in \Phi_{\alpha[j]}$ und eine jüngste Aktion $\alpha[l] \in W(\alpha_{j-1})$ existieren, $1 \leq l < j$, $v_{i_j} \in SV_{\alpha[l]}$, so daß gilt:

$$(*)\ v_{i_j}[val_P(\alpha_l)] \neq v_{i_j}[val_P(r_1(\alpha_l))]$$

Da $r_1(\alpha[l]) \neq \lambda$ sein muß, andernfalls würde $\alpha[j]B_a\alpha[l]$ und damit $z(\alpha[j])$=**true** gelten im Widerspruch zur Annahme $r_1(\alpha[j]) \neq \lambda$, muß eine weitere globale Variable $v_{i_l} \in \Phi_{\alpha[l]}$ existieren, für die $v_{i_j}[val_P(\alpha_{l-1})] \neq v_{i_l}[val_P(r_1(\alpha_{l-1}))]$ gilt.
Definiert man $j=l$, so läßt sich letztere Folgerung fortführen bis schließlich $l=0$ erreicht wird. Da in diesem Falle für alle $v \in V_P$ gilt:

$$v[val_P(\lambda)]=v[val_P(r_1(\lambda))],$$

erhält man einen Widerspruch zu (*).

Fall 2 ($l(v)$=**true** & $\neg\exists\alpha[i]\subset W(\alpha)[v\in V_{\alpha[i]}$ & $r_1(\alpha[i])=\lambda]$): Nach Definition von r_1 gilt in diesem Fall:

$$v[r_1(val_P(\alpha))]::=v[val_P(\alpha)].$$

Zu zeigen bleibt, daß $v[val_P(r_1(\alpha))]$ gleich $v[val_P(\alpha)]$ ist. Dies läßt sich mit einem zu Fall 1 analogen Beweis nachweisen.

Fall 3 ($g(v)$=**true** & $v\in SV_{\alpha[i]}$ & $r_1(\alpha[i])\neq\lambda$ & i maximal): Da für eine lokale Variable v' $v'[r_1(val_P(\alpha))]\neq v'[val_P(r_1(\alpha))]$ nicht gelten kann (siehe Fall 1 und 2), läßt sich die Annahme $v[r_1(val_P(\alpha))]\neq v[val_P(r_1(\alpha))]$ auch im Falle 3 mit der gleichen Argumentation wie am Ende von Fall 1 auf einen Widerspruch führen.

Fall 4 ($g(v)$=**true** & $\neg\exists\alpha[i]\in W(\alpha)$: $[v\in SV_{\alpha[i]}$ & $r_1(\alpha[i])\neq\lambda]$): In diesem Falle gilt $v[r_1(val_P(\alpha))]=v[val_P(\lambda)]$ und, da für alle $\alpha[i]\in W(\alpha)$ mit der Eignschaft $v\in SV_{\alpha[i]}$ $r_1(\alpha[i])=\lambda$ gilt, ist auch $v[val_P(r_1(\alpha))]=v[val_P(\lambda)]$ im Widerspruch zur Annahme:

$$v[val_P(r_1(\alpha))]\neq v[r_1(val_P(\alpha))].$$

Mit Fall 4 ist die vollständige Fallunterscheidung und damit der Nachweis von

$$val_P(r_1(\alpha))=r_1(val_P(\alpha))$$

abgeschlossen. Bleibt zu zeigen, daß $v[r_1(val_P(\alpha))]$ (potentiell) fehlerfrei ist.

(3) Nachweis der (potentiellen) Fehlerfreiheit von $r_1(val_P(\alpha))$: Da $r_1(\alpha)$, wie unter (1) bewiesen, aktiv ist und

$$\bigcup_{\alpha[i]\in W(r_1(\alpha))\&f_{r_1(\alpha)}(\alpha[i])} Z_{r_1(\alpha)}(\alpha[i]) = \phi$$

(letzteres deshalb, weil alle fehlerhaften und zurückzusetzenden Aktionen innerhalb der Berechnung α nach Definition der Recoveryfunktion r_1 zurückgesetzt wurden), ist $val_P(r_1(\alpha))$ (potentiell) fehlerfrei (vgl. Hilfssatz 3.3.4/1).
Da, wie unter (2) bewiesen, $val_P(r_1(\alpha))=r_1(val_P(\alpha))$ gilt, folgt damit auch die (potentielle) Fehlerfreiheit von $r_1(val_P(\alpha))$. ■

Im Satz 3.4.2/1 wurde die Recoveryfunktion r_1 für den Fall zurückzusetzender Aktionen $\alpha[i]$ definiert. Der folgende Hilfssatz 3.4.3/1 kann dahingehend interpretiert werden, daß n Fehler dadurch korrigiert werden können, in dem sie einzeln nacheinander behoben werden, d.h. *Mehrfachfehler* können durch eine entsprechende Anzahl von *Einzelfehlerkorrekturen* beseitigt werden. Dies gilt sowohl für den Fall der Nichtberücksichtigung als auch, was mit einer zu Hilfssatz 3.4.2/2 analogen Aussage gezeigt werden kann, den der Berücksichtigung potentieller Fehlerverursacher.

Hilfssatz 3.4.2/2: Sei α eine aktive Berechnungsfolge eines Prozeßsystems $P=(A,D,w)$. Dann gilt:

$$\{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\} \;=\; \bigcup_{\alpha[i] \in W(\alpha) \;\&\; f_\alpha(\alpha[i])} Z_\alpha(\alpha[i]).$$

Beweis: Es genügt zu zeigen, daß gilt:

$$(1) \qquad \forall \alpha[i] \in W(\alpha): [f_\alpha(\alpha[i]) = \textbf{true} \Rightarrow Z_\alpha(\alpha[i]) \subseteq \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}]$$

und

$$(2) \qquad \bigcup_{\alpha[i] \in W(\alpha) \;\&\; f_\alpha(\alpha[i])} Z_\alpha(\alpha[i]) \supseteq \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}.$$

zu(1): Angenommen $Z_\alpha(\alpha[i]) \subseteq \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}$ würde nicht gelten. Dann müßte ein $\alpha[k] \in Z_\alpha(\alpha[i])$ existieren, für das gilt:

$$\alpha[k] \notin \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}.$$

$\alpha[k] \in Z_\alpha(\alpha[i])$ ist nach Hilfssatz 3.2.4/2 genau dann erfüllt, wenn $z_\alpha(\alpha[k]) = \textbf{true}$. Damit folgt $\alpha[k] \in \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}$, im Widerspruch zur Annahme $\alpha[k] \notin \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}$.

zu(2): Angenommen

$$\bigcup_{\alpha[i] \in W(\alpha) \;\&\; f_\alpha(\alpha[i])} Z_\alpha(\alpha[i]) \supseteq \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}$$

würde nicht gelten. Dann müsste es ein

$$\alpha[k] \in \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}$$

geben, das in $\bigcup_{\alpha[i] \in W(\alpha) \;\&\; f_\alpha(\alpha[i])} Z_\alpha(\alpha[i])$ nicht enthalten ist.
$\alpha[k] \in \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; z_\alpha(\alpha[j]) = \textbf{true}\}$ bedeutet jedoch nach Hilfssatz 3.2.4/1:

$$f_\alpha(\alpha[k]) \;\|\; \exists\, \alpha[i] \in W(\alpha): [z_\alpha(\alpha[i]) = \textbf{true} \;\&\; \alpha[k](E_\alpha \cup B_\alpha)\alpha[i]]$$

Zwei Fälle sind zu unterscheiden:

Fall 1 ($f_\alpha(\alpha[k]) = \textbf{true}$): In diesem Falle gilt

$$\alpha[k] \in Z_\alpha(\alpha[k]) \subseteq \bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} Z_\alpha(\alpha[i])$$

im Widerspruch zur Annahme.

Fall 2 ($z_\alpha(\alpha[i])$=**true** & $\alpha[k](B_\alpha \cup E_\alpha)\alpha[i]$) Nach Hilfssatz 3.2.4/2 existiert ein $\alpha[l]$, so daß $\alpha[i] \in Z_\alpha(\alpha[l])$ & $f_\alpha(\alpha[l])$=**true**. Nach Definition von $Z_\alpha(\alpha[l])$ folgt damit

$$\alpha[k](B_\alpha \cup E_\alpha)\alpha[i](B_\alpha \cup E_\alpha)^*\alpha[l],$$

d.h.

$$\alpha[k] \in Z_\alpha(\alpha[l]) \subseteq \bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} Z_\alpha(\alpha[i])$$

gilt auch in diesem Falle im Widerspruch zur Annahme, und damit ist der Hilfssatz 3.4.2/2 bewiesen. ■

Es ist die Recoveryfunktion r_1, die in den Arbeiten von K.H. Kim, [Kim78a], [Kim78b], [Kim78c], [Kim79], [Kim80], K. Kant, [Kan81], und D.L. Russell, [Rus76], [Rus77], [Rus80], implizit zugrundegelegt wird, ohne näher definiert zu werden.

3.4.3. Koordinierungsbezogene Recoveryfunktion

Im Gegensatz zu der im vorangehenden Abschnitt vorgestellten Recoveryfunktion r_1, die geeignet ist jede relevante Fehlersituation zu korrigieren, ist die nachfolgend definierte Recoveryfunktion r_2 nur in besonderen Fehlersituationen, wie sie in Synchronisationssystemen vorkommen können, anwendbar. Motiviert wird ihre Definition durch den geringeren Fehlerbehandlungsaufwand (gemessen in der Zahl zurückzusetzender Aktionen), den sie im Vergleich zu r_1 verursacht, da sie in einer Fehlersituation, in der sie anwendbar ist, nicht zwangsläufig alle Aktionen $\alpha[i]$ einer fehlerhaften Berechnungsfolge α mit der Eigenschaft $z_\alpha(\alpha[i])$=**true** zurücksetzt, um zu einem (potentiell) fehlerfreien Prozeßsystem-Zustand zu gelangen.
In diesem Zusammenhang ist zu beachten, daß sich die Aussage von Hilfssatz 3.2.4/1 auf den allgemeinen Fall eines beliebigen Prozeßsystems $P=(A,D,w)$, einer beliebigen Berechnung α von P und einer nicht näher bekannten (!) Fehlersituation bezieht. Das fehlende Wissen um eine spezielle Fehlersituation führt dazu, daß in einer gegebenen Fehlersituation möglicherweise mehr Aktionen als unbedingt erforderlich als zurückzusetzende Aktionen eingestuft werden können. Die Existenz von Recoveryfunktion r_2 steht also nicht im Widerspruch zur Aussage des Hilfssatzes 3.2.4/1.

Bei der Definition der Recoveryfunktion r_2 wird, vereinfacht ausgedrückt, versucht den Umstand auszunützen, daß, falls zwischen zwei zu verschiedenen Prozessen gehörenden Synchronisationsaktionen $\alpha[i]$ und $\alpha[j]$ einer Berechnung α ein Abhängigkeitsverhältnis der Art $\alpha[j]B_\alpha\alpha[i]$ besteht, bei Zurücksetzung von $\alpha[i]$ die

Aktion $\alpha[j]$ nur dann zurückgesetzt werden muß, wenn ohne die Ausführung von $\alpha[i]$ innerhalb von α die Aktion $\alpha[j]$ nicht lauffähig wäre, vorausgesetzt die Rekonstruktion des zugehörigen konsistenten Prozeßsystem-Zustandes ist gesichert.

Beispiel 3.4.3/1: Sei $P=(A,D,w)$ ein Synchronisationssystem mit den Aktionen $a_{1,1}$: $V(S)$; und $a_{2,1}$: $P(S)$. Im Anfangszustand w gelte. $S[w]=1$, $bz_1[w]=adr(a_{1,1})$ und $bz_2[w]=adr(a_{2,1})$. Dann ist die Aktionenfolge $a_{1,1}a_{2,1}$ aktiv in P und $a_{2,1}$ ist auch ohne die Ausführung von $a_{1,1}$ aktiv. Im Zustand $val_P(a_{1,1}a_{2,1})$ kann demzufolge trotz der Beziehung $a_{2,1}B_{a_{1,1}a_{2,1}}a_{1,1}$ prinzipiell die Aktion $a_{1,1}$ zurückgesetzt werden, ohne daß $a_{2,1}$ ebenfalls zurückgesetzt werden muß. Die Rekonstruktion von $val_P(a_{2,1})$ ist ausgehend vom Zustand $val_P(a_{1,1}a_{2,1})$ möglich, da zur Operation $S=S+1$ der Aktion $a_{1,1}$ eine *inverse* Operation $S=S-1$ existiert, mit welcher der Wert von S ausgehend von $val_P(a_{1,1}a_{2,1})$ entsprechend, ohne Rücksetzung in den Zustand $val_P(\lambda)$, errechnet werden kann.

Satz 3.4.3/1: Sei $P=(A,D,w)$ ein Synchronisationssystem, und für alle Variablen $v \in V_P$ gelte $l(v)=$**true** $\parallel s(v)=$**true**. Für *alle* Berechnungsfolgen α, β, γ von P gelte (eine Variable $v \in V_P$, für die $s(v)=$**true** gilt, wird im folgenden mit S bezeichnet, wenn verdeutlicht werden soll, daß v eine Semaphorvariable ist):

(1) $[\forall S \in V_P: [S[val_P(\alpha)] \le S[val_P(\beta)] \ \& \ \alpha$ aktiv & β aktiv$]] \Rightarrow$
$\Rightarrow [\alpha\gamma$ aktiv $\Rightarrow \beta\gamma$ aktiv$]$ (*Monotonie-Eigenschaft*).

Des weiteren gelte für alle aktiven Berechnungen α von P (mit $\Theta(\alpha[j]|S)$ wird im folgenden die Größe der Änderung von S durch $\alpha[j]$ bezeichnet):

(2) $$\forall S \in V_P: [S[val_P(\alpha)] = S[val_P(\lambda)] + \sum_{j=0}^{|\alpha|} \Theta(\alpha[j] \mid S)].$$

Sei α aktiv in P, $\alpha[i]$ eine fehlerhafte Aktion, die zurückgesetzt werden muß, und es gelte:

(3) $$\exists k<i: [\forall S \in V_{P_{bz(\alpha[i])}} \forall l \in \{k,\ldots,|\alpha|\}: \sum_{\alpha[j] \in M_{k,l}} \Theta(\alpha[j] \mid S) \le 0],$$

wobei $M_{k,l} ::= W(\alpha \mid [t(\alpha[k]),t(\alpha[l])]) \cap P_{bz(\alpha[i])}$ und für $\alpha[k]$ bzw. k gilt: $((wp(\alpha[k])=$**true** $\& \ \alpha[k] \in P_{bz(\alpha[i])}) \ \parallel \ k=0)$.

Dann ist die wie folgt definierte Funktion r_2 *Recoveryfunktion* zu α:

$$r_2(\lambda) ::= \lambda, \ r_2(\alpha) ::= r_2(\alpha[1])\ldots r_2(\alpha[|\alpha|]),$$

$$r_2(\alpha[j]) ::= \begin{cases} \lambda & \alpha[j] \in M_{k,|\alpha|} \\ \quad falls & \\ \alpha[j] & sonst \end{cases}$$

wobei $j \subset \{1,\ldots,|\alpha|\}$ und k maximal bzgl. Eigenschaft (3) gewählt sei.

Beweis: Es muß gezeigt werden, daß gilt:

(1) $r_2(\alpha)$ *aktiv*,

(2) $val_P(r_2(\alpha))$ von $val_P(\alpha)$ ausgehend rekonstruierbar und

(3) $\bigcup_{\alpha[j] \in W(r_2(\alpha)) \& f_{r_2(\alpha)}(\alpha[j])} Z_{r_2(\alpha)}(\alpha[j]) = \phi$

zu(:) Da sich alle nichtsynchronisierenden Aktionen nur auf lokale Variablen beziehen (für alle $v \in V_P$ gilt: $l(v)$=**true** || $s(v)$=**true**), können nichtsynchronisierende Aktionen anderer Prozesse durch die Zurücksetzung von Aktionen des Prozesses $P_{bz(\alpha[i])}$ nicht direkt betroffen sein. Daher kann im folgenden o.B.d.A. angenommen werden, daß alle Aktionen $a \in W(\alpha)$ synchronisierende Aktionen sind.
Angenommen $r_2(\alpha)$ wäre nicht aktiv. Dann muß es ein j, $0<j\leq|\alpha|$, geben, so daß gilt: α_j aktiv, $r_2(\alpha_{j-1})$ aktiv und $r_2(\alpha_{j-1})r_2(\alpha[j])$ nicht aktiv ($r_2(\alpha[j])$ ist in diesem Fall zwangsläufig ungleich λ, sonst wäre $r_2(\alpha_{j-1})r_2(\alpha[j])$ aktiv). Das heißt aber, daß - aufgrund der Definition einer synchronisierenden Aktion (vgl. Definition 2.1.2.4/1-(2)) und der Voraussetzung (1) - für eine Semaphorvariable $S \in L_{\alpha[j]}$ gelten muß:

$$(*) \quad S[val_P(\alpha_{j-1})] > S[val_P(r_2(\alpha_{j-1}))].$$

Da jedoch (nach Voraussetzung (2)):

$$S[val_P(\alpha_{j-1})] = S[val_P(\lambda)] + \sum_{k=0}^{j-1} \Theta(\alpha[k] \mid S),$$

und nach Voraussetzung (3) ein l existiert, so daß gilt:

$$S[val_P(\alpha_{j-1})] = S[val_P(\lambda)] + \sum_{k=0}^{j-1} \Theta(\alpha[k] \mid S) =$$
$$= S[val_P(\lambda)] + \sum_{k=0}^{j-1} \Theta(r_2(\alpha[k] \mid S) + \sum_{\alpha[k] \in M_{l,j-1}} \Theta(\alpha[k] \mid S),$$

sowie nach Definition von r_2:

$$S[val_P(r_2(\alpha_{j-1}))] = S[val_P(\lambda)] + \sum_{k=0}^{j-1} \Theta(r_2(\alpha[k])| S)$$

kann, da nach Voraussetzung (3):

$$\sum_{\alpha[k] \in M_{l,j-1}} \Theta(\alpha[k]\,; S) \leq 0,$$

auf $S[val_P(\alpha_{j-1})] \leq S[val_P(r_2(\alpha_{j-1}))]$ geschlossen werden im Widerspruch zu (*) und damit der Annahme $r_2(\alpha)$ sei nicht aktiv

zu(2): Im ersten Schritt wird gezeigt, daß ausgehend vom Prozeßsystem-Zustand $val_P(\alpha)$ ein Prozeßsystem-Zustand $r_2(val_P(\alpha))$ rekonstruiert werden kann, von dem im zweiten Schritt nachgewiesen wird, daß er mit $val_P(r_2(\alpha))$ übereinstimmt.

Schritt 1: Im Synchronisationssystem P kommen zwei Typen von Variablen vor: Semaphorvariablen und lokale Variablen $v \in V_P$.
Für lokale Variablen $v \in V_P$ wird ein - wie später bewiesen - konsistenter Zustand wie folgt rekonstruiert: Ist $\alpha[j]$, $j \leq |\alpha|$, die älteste Aktion des Prozesses $P_{bz(\alpha[i])}$, für die $r_2(\alpha[j])=\lambda$ gilt, und gehört v zu Prozeß $P_{bz(\alpha[i])}$, so kann mit Hilfe des Wiederaufsetzpunktes $WP_a(\alpha[j])$ in bezug auf alle (lokalen) Variablen $v \in V_P \cap V_{P_{bz(\alpha[i])}}$ zu Prozeß $P_{bz(\alpha[i])}$ der Prozeßsystem-Zustand $val_P(\alpha_{j-1})$ wiederhergestellt werden.
Bzgl. der Semaphorvariablen $v \in V_P$ kann mit Hilfe der zu diesen Variablen existierenden Variablenhistorie (Semaphorvariablen werden als globale Variablen betrachtet) der folgende Zustand:

$$v[val_P(\alpha)] - \sum_{\alpha[k] \in M_{l,|\alpha|}} \Theta(\alpha[k] \mid v)$$

rekonstruiert werden, wobei l bzgl. (3) maximal gewählt sei.
Mit diesen Erkenntnissen läßt sich $v[r_2(val_P(\alpha))]$ für alle $v \in V_P$ des Synchronisationssystems $P=(A,D,w)$ bzgl. der aktiven Berechnungsfolge α wie folgt definieren (j sei minimal bzgl. der Eigenschaft $r_2(\alpha[j])=\lambda$ und $\alpha[j]$ gehöre zu $P_{bz(\alpha[i])}$):

$$v[r_2(val_P(\alpha))] ::= \begin{cases} v[val_P(\alpha)] - \sum_{\alpha[k] \in M_{l,|\alpha|}} \Theta(\alpha[k] \mid v) & s(v) \\ v[val_P(\alpha_{j-1})] & falls \ \neg s(v) \ \& \ v \in V_{P_{bz(\alpha[i])}} \\ v[val_P(\alpha)] & sonst \end{cases}$$

Schritt 2: Zu zeigen bleibt, daß für alle $v \in V_P$ gilt:

$$v[r_2(val_P(\alpha))] \ = \ v[val_P(r_2(\alpha))].$$

Beim Nachweis dieser Beziehung werden drei Fälle unterschieden:

Fall 1 ($s(v)$=**true**): Da nach Voraussetzung (2) und nach Wahl von l gilt:

$$v[val_P(\alpha)] \ = \ v[val_P(\lambda)] + \sum_{k=0}^{|\alpha|} \Theta(\alpha[k] \mid v) =$$

$$= v[val_P(\lambda)] + \sum_{k=0}^{|\alpha|} \Theta(r_2(\alpha[k])|v) + \sum_{\alpha[k] \in M_{i,|\alpha|}} \Theta(\alpha[k] \mid v),$$

und.

$$v[val_P(r_2(\alpha))] = v[val_P(\lambda)] + \sum_{k=0}^{|\alpha|} \Theta(r_2(\alpha[k]) \mid v),$$

folgt:

$$v[val_P(r_2(\alpha))] = v[val_P(\alpha)] - \sum_{\alpha[k] \in M_{i,|\alpha|}} \Theta(\alpha[k] \mid v),$$

und damit:

$$v[val_P(r_2(\alpha))] = v[r_2(val_P(\alpha))].$$

Fall 2 ($\neg s(v)$ & $v \in V_{P_{bz(\alpha[i])}}$): Da v keine Semaphorvariable ist ($\neg s(v)$), ist v nach Voraussetzung eine lokale Variable. Nach Wahl von j gilt $r_2(\alpha_{j-1}|P_{bz(\alpha[i])}) = (\alpha_{j-1}|P_{bz(\alpha[i])})$ und daher auch $v[val_P(r_2(\alpha_{j-1}))] = v[val_P(\alpha_{j-1})]$ ($\alpha[j]$ ist die älteste Aktion des Prozesses $P_{bz(\alpha[i])}$, die zurückgesetzt werden muß).
Da des weiteren nach Definition von r_2 $W(r_2(\alpha))[t(\alpha[j]), t(\alpha[|\alpha|])] \cap P_{bz(\alpha[i])} = \phi$, kann v als lokale Variable durch kein $r_2(\alpha[k])$, $j < k \leq |\alpha|$, verändert werden. Daher gilt:

$$v[val_P(r_2(\alpha))] = v[val_P(\alpha_{j-1})] = v[r_2(val_P(\alpha))].$$

Fall 3 ($\neg s(v)$ & $v \notin V_{P_{bz(\alpha[i])}}$): Da nach Definition von r_2 (es werden nur Aktionen des Prozesses $P_{bz(\alpha[i])}$ zurückgesetzt) die Ausführungsreihenfolge in $r_2(\alpha)$ in bezug auf Aktionen, welche die lokalen Variablen $v \in V_P - V_{P_{bz(\alpha[i])}}$ verändern, mit derjenigen in α übereinstimmt, folgt:

$$v[val_P(r_2(\alpha))] = v[val_P(\alpha)] = v[r_2(val_P(\alpha))].$$

zu(3): Da nach Voraussetzung nur $\alpha[i]$ als fehlerhaft klassifiziert wurde und nach Definition von r_2: $r_2(\alpha[i]) = \lambda$, gilt für alle $\alpha \in W(r_2(\alpha))$: $f_{r_2(\alpha)}(\alpha) \neq \mathbf{true}$. Damit ist die Menge

$$\bigcup_{\alpha[j] \in W(r_2(\alpha)) \;\&\; f_{r_2(\alpha)}(\alpha[j])} Z_{r_2(\alpha)}(\alpha[j]) =$$
$$\{\alpha[k] \mid \alpha[k] \in W(r_2(\alpha)) \;\&\; \exists\, \alpha[j] \in W(r_2(\alpha)) \colon [\alpha[k](B \cup E)^* \alpha[j] \;\&\; f_{r_2(\alpha)}(\alpha[j])]\}$$

leer

$$\bigcup_{\alpha[j] \in W(r_2(\alpha)) \& f_{r_2(\alpha)}(\alpha[j])} Z_{r_2(\alpha)}(\alpha[j]) = \phi$$

ist hiermit für alle $\alpha[j] \in W(r_2(\alpha))$ nachgewiesen. ■

Korollar 3.4.3/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Für P gelte $s(P)$=**true**, d.h. in aktiven Berechnungen α von P sollen nur koordinierungsbedingte Abhängigkeiten bestehen. Besitzen *alle* Berechnungsfolgen von P die in Satz 3.4.3/1 skizzierten Eigenschaften (1), (2), und ist $\alpha[i]$ eine fehlerhafte Aktion einer aktiven Berechnung α von P mit Eigenschaft (3), dann ist die in Satz 3.4.3/1 beschriebene Funktion r_2 Recoveryfunktion zu α.

Beweis: Es genügt zu zeigen, daß von $s(P)$=**true** o.B.d.A. auf

$$(*) \quad \forall v \in V_P\colon [l(v)=\textbf{true} \parallel s(v)=\textbf{true}]$$

geschlossen werden kann. Der Rest des Korollars ist dann eine unmittelbare Folge von Satz 3.4.3/1.
Die Zulässigkeit der Schlußfolgerung $s(P) \Rightarrow o\,B\,d.A.$ (*) wird wie folgt nachgewiesen: Zu $P=(A,D,w)$ wird ein Prozeßsystem $P'=(A',D',w')$ konstruiert, daß:

(i) die vorausgesetzten Eigenschaften (1) bis (3) des Satzes 3.4.3/1 besitzt und

(ii) für das $s(P')$=**true** sowie

(iii) α aktiv in P gdw. α' aktiv in P' gilt.

P' wird ausgehend von P nach folgender Konstruktionsvorschrift abgeleitet. Jede Variable v_i eines jeden Prozesses P_j von P mit der Eigenschaft $s(v_i)$=**false** wird durch eine Variable $v'_{j,i}$ ersetzt, $v'_{j,i}[val_{P'}(\lambda)] = v_i[val_P(\lambda)]$, für $v'_{j,i}$ gilt $l(v'_{j,i})$=**true**. Sonst sei P' gegenüber P unverändert.
Für das nach dieser Konstruktionsvorschrift erhaltene Prozeßsystem P' gilt offensichtlich:

$$(*)' \quad \forall v' \in V_{P'}\colon [l(v')=\textbf{true} \parallel s(v')=\textbf{true}].$$

Für P kann nun o.B.d.A. auf (*) geschlossen werden, sobald die Gültigkeit von (i)-(iii) für P' nachgewiesen ist.
Aufgrund der Eigenschaft (*)' von P' ist (ii) trivialerweise erfüllt, und es ist leicht zu sehen, daß die Gültigkeit von (i) für P' gezeigt ist, sobald (iii) nachgewiesen ist.
Für den Nachweis von (iii) genügt es nun zu zeigen, daß gilt:

$$(**) \quad \neg(\exists\alpha\,\exists\alpha'\,\exists k\,\exists v_i\,\exists j\ [(\alpha \textit{ aktiv} \not\Leftrightarrow \alpha' \textit{ aktiv}) \;\&\; v_i[val_P(\alpha_k)] \neq v'_{j,i}[val_{P'}(\alpha'_k)] \;\&\; v'_{j,i} \in LV_{\alpha'[k+1]} \;\&\; k \textit{ minimal}]).$$

Angenommen (**) würde nicht gelten. Dann sind zwei Fälle zu unterscheiden:

Fall 1 ($\neg\exists\alpha'[l]\colon [v'_{j,i} \in SV_{\alpha'[l]} \;\&\; l<k+1]$). In diesem Fall kann es, aufgrund der skizzierten Konstruktionsvorschrift zu P', auch kein $v_i \in SV_{\alpha[l]}$ geben, d.h. aber:

$$v_i\ val_P(\alpha_k)]=v_i[val_P(\lambda)]=v'_{j,s}[val_{P'}(\lambda)]=v'_{j,s}[val_{P'}(\alpha'_k)]$$

im Widerspruch zur Annahme ¬(**).

Fall 2 ($\exists \alpha'[l]\ v'_{j,s}\in SV_{\alpha'[l]}$ & $l<k+1$]) Wähle l maximal. Aufgrund der Wahl von l muß gelten $\alpha[k+1]B_a\alpha[l]$
Da nach Annahme $v_i\in V_{\alpha[k+1]}$ und $s(v_i)$=**false**, muß, falls α aktiv ist wegen $s(P)$=**true**, falls α' aktiv ist wegen $s(P')$=**true**, $l(\alpha[k+1]B_a\alpha[l])$=**true** (bzw. $l(\alpha'[k+1]B_a\alpha'[l])$=**true**) gelten, d.h. $proc_P(\alpha[k+1],\alpha[l])$ und damit folgt:

$$j=bz(\alpha[k+1])=bz(\alpha[l]).$$

Da aufgrund der Maximalität von l gilt:

$$v_i[val_P(\alpha_l)]=v_i[val_P(\alpha_k)]\ \&\ v'_{j,s}[val_{P'}(\alpha'_l)]=v'_{j,s}[val_{P'}(\alpha'_k)],$$

genügt es $v'_{j,s}[val_{P'}(\alpha'_l)]=v_i[val_P(\alpha_l)]$ zu zeigen, um die Annahme ¬(**) auf einen Widerspruch zu führen.
Zum Nachweis von $v'_{j,s}[val_{P'}(\alpha'_l)]=v_i[val_P(\alpha_l)]$ genügt es zu zeigen:

(***) $\forall v_m\in LV_{\alpha[l]}$: $[(\alpha\ aktiv\ \|\ \alpha'\ aktiv)\ \&\ v_m[val_P(\alpha_{l-1})]=v'_{j,m}[val_{P'}(\alpha'_{l-1})]]$.

Da, würde (***) nicht gelten, dies ein Widerspruch zur Minimalität von k in (**) wäre (beachte setzt man $k=l-1$, so sind (**) und (***), bis auf die Forderung k sei minimal, vom Aussagewert her identisch), denn nach Annahme gilt $l<k+1$, muß (***) und damit auch $v'_{j,s}[val_{P'}(\alpha'_l)]=v_i[val_P(\alpha_l)]$ gelten.
Damit ist die Zulässigkeit der Folgerung $s(P)$ => *o.B.d.A* (*) und, unter Rückgriff auf die Aussage von Satz 3.4.3/1, auch Korollar 3.4.3/1 bewiesen. ■

Wie ein Vergleich der Definitionen von r_1 (vgl. Abschnitt 3.4.2) und r_2 zeigt, ist es neben Eigenschaft (2) vor allem die Eigenschaft (1), welche die Recoveryfunktion r_2 gegenüber r_1 auszeichnet:

(1) r_1 realisiert ein Fehlerbehandlungsverfahren, das vom Prinzip her einem *Rückwärtsrecoveryverfahren*, [AL81], entspricht. Das durch r_2 beschriebene Verfahren kann hingegen als ein *Vorwärtsrecoveryverfahren*, [AL81], bezeichnet werden.

(2) Beim Auftreten eines Fehlers innerhalb eines Prozesses P_i werden, die Anwendbarkeit von r_2 vorausgesetzt, zur Rekonstruktion eines (potentiell) fehlerfreien Zustandes nur Aktionen von P_i zurückgesetzt, obwohl Abhängigkeiten zu anderen Prozessen bestehen können, die bei Anwendung von r_1 auch zur Zurücksetzung von Aktionen dieser Prozesse führen würden.

Zu (1) ist zu bemerken, daß in der Literatur Vorwärts- und Rückwärtsrecoveryver-

fahren anhand des folgenden Kriteriums unterschieden werden [AL81], [Mil85]. Ein Fehlerbehandlungsverfahren wird als *Rückwärtsrecoveryverfahren* (*Backward Error Recovery*) bezeichnet, wenn der mittels dieses Verfahrens rekonstruierte Wiederaufsetzzustand ausschließlich unter Rückgriff auf gesicherte Wiederaufsetzpunkte abgeleitet wurde. Andernfalls trägt das Fehlerbehandlungsverfahren Merkmale eines *Vorwärtsrecoveryverfahrens* (*Forward Error Recovery*).
Die Recoveryfunktion r_2 realisiert in diesem Sinne ein Vorwärtsrecoveryverfahren auf Semaphorvariablen S, da sie durch Subtraktion des Wertes $\sum_{\alpha[k]\in M_{1,|\alpha|}} \Theta(\alpha[k]|S)$, vom aktuellen Zustand $S[val_P(\alpha)]$ einer Berechnung α ausgehend, den Wiederaufsetzzustand $S[r_2(val_P(\alpha))]$ ermittelt.
Zu (2) ist zu sagen, daß sich dieser Unterschied zwischen r_1 und r_2 am besten am Beispiel einer häufig in realen Systemen vorkommenden Synchronisationsbeziehung zwischen zwei Prozessen demonstrieren läßt:

Beispiel 3.4.3/2: Das in Bild 3.4.3/1 skizzierte Petri-Netz stellt ein Prozeßsystem $P=(A,D,w)$ dar, in dem die Folge

$$\alpha ::= a_{2,1}a_{2,2}a_{2,3}a_{2,4}(a_{1,1}a_{1,2}a_{1,3}a_{1,4})^*a_{1,1}a_{2,5}$$

aktiv ist. Wird die Aktion $a_{2,5}$ in α als fehlerhaft erkannt, so errechnen die Recoveryfunktionen r_1 und r_2 die folgenden Wiederaufsetzzustände:

$r_1(val_P(\alpha))=val_P(a_{2,1})$,
$r_2(val_P(\alpha))=val_P(a_{2,1}(a_{1,1}a_{1,2}a_{1,3}a_{1,4})^*a_{1,1})$

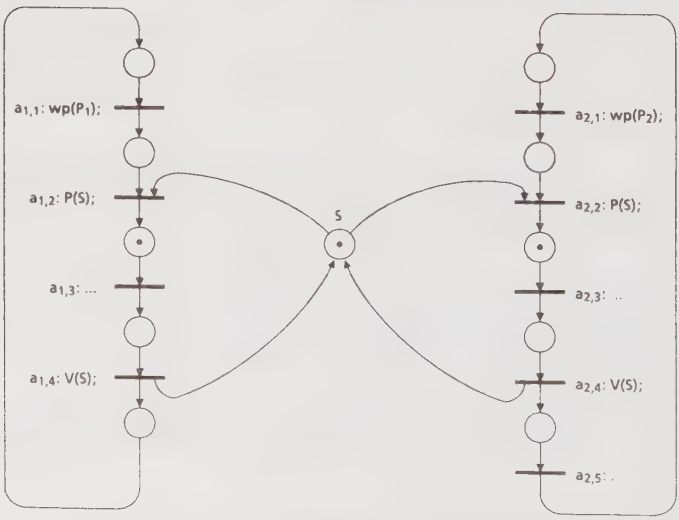


Bild 3.4.3/1: Typische Synchronisationsbeziehung zwischen zwei Prozessen

Von praktischer Bedeutung sind Recoveryfunktionen wie r_2 insbesondere dann, wenn Prozesse nur zum Zwecke der Koordinierung des Zugriffs auf gemeinsame Betriebsmittel miteinander kommunizieren, im eigentlichen Sinne aber keine Information austauschen, d h Betriebsmittel nach Benutzung mit anschließender Freigabe z.B. in ein und denselben (definierten) Anfangszustand versetzt werden.
Derartige Ideen werden von S K Shrivastava und J Banatre in [SB78] diskutiert, und es wird von ihnen ein *Port-Mechanismus* eingeführt, über den der Zugriff auf Betriebsmittel die kein Gedächtnis besitzen, koordiniert werden kann, und der für den Fehlerfall effiziente Fehlerbehandlungsmaßnahmen vorsieht.

3.4.4. Kommunikationsbezogene Recoveryfunktion

Ähnlich wie für Systeme koordinierender Prozesse, läßt sich in bestimmten Fehlersituationen auch für Systeme kommunizierender Prozesse eine Recoveryfunktion r_3 definieren, welche sich gegenüber der Recoveryfunktion r_1 in den beiden folgenden Punkten auszeichnet.

(1) r_3 realisiert ein *Vorwärtsrecoveryverfahren*,

(2) in den Situationen, in welchen r_3 anwendbar ist, beschränkt sich die Korrektur eines Fehlers innerhalb eines Prozesses P_i ausschließlich auf P_i selbst.

Die Idee zur Definition von r_3 leitet sich von dem Wunsch ab, den Empfang von Nachrichten durch einen Prozeß P_i im Fehlerfall gegebenenfalls durch Rückgabe dieser Nachrichten rückgängig zu machen, ohne dabei den Fortgang anderer Prozesse zu beeinflussen.

Satz 3.4.4/1: Sei $P=(A,D,w)$ ein *RANDOM*-Nachrichtensystem. α sei eine aktive Berechnung von P. $\alpha[i]$ sei eine fehlerhafte Aktion, die zurückgesetzt werden muß, und es gelte:

(1) $\forall \alpha[j] \in P_{bz(\alpha[i])}: [_{vl}(\alpha[j]) => \alpha[j]:receive_{RANDOM}(\ldots,\ldots)]$.

Für Nachrichtenvariable nv_j des Prozesses $P_{bz(\alpha[i])}$ gelte:

(2) $\forall nv_j \in V_{P_{bz(\alpha[i])}}: [l(nv_j)=\mathbf{true}]$.

Des weiteren gelte für alle Nachrichtenpuffer np_i, auf die von $P_{bz(\alpha[i])}$ aus zugegriffen werden kann:

(3) $\forall np_i \in V_{P_{bz(\alpha[i])}}: [f_i[val_P(\lambda)]=\infty]$.

Dann ist die wie folgt definierte Funktion r_3 Recoveryfunktion zu α:

$$r_3(\alpha[j]) = \begin{cases} \lambda & \alpha[j] \in M_k \\ & falls \\ \alpha[j] & sonst \end{cases}$$

wobei $k<i$ maximal gewählt sei bzgl. der Eigenschaft

$$wp(\alpha[k])=\textbf{true} \;\&\; \alpha[k] \in P_{bz(\alpha[i])}$$

(falls so ein k nicht existiert, sei $k=0$) und

$$M_k ::= W(\alpha \mid [t(\alpha[k]), t(\alpha[|\alpha|])]) \cap P_{bz(\alpha[i])}.$$

Beweis: Der Beweis dieses Satzes kann, unter Ausnutzung der Eigenschaften (1) bis (3) und der Tatsache, daß es sich um ein *RANDOM*-Nachrichtensystem handelt, nach dem gleichen Schema erfolgen wie die Beweise der Sätze 3.4.2/1 und 3.4.3/1. Aus diesem Grunde wird auf einen ausführlichen Beweis an dieser Stelle verzichtet. Es wird lediglich skizziert wie, vom Zustand $val_P(\alpha)$ ausgehend, der Zustand $r_3(val_P(\alpha))$ rekonstruiert werden kann, der, wie sich zeigen läßt, mit dem Zustand $val_P(r_3(\alpha))$ übereinstimmt.

Für alle Nachrichtenpuffer $np_l \in V_{P_{bz(\alpha[i])}}$, für die $g(np_l)=\textbf{true}$ gilt und zu welchen Aktionen des Prozesses $P_{bz(\alpha[i])}$ Zugriff besitzen, lassen sich $r_3(np_l[val_P(\alpha)])$, $r_3(b_l[val_P(\alpha)])$ und $r_3(f_l[val_P(\alpha)])$ nach folgender Vorschrift ermitteln:

$$r_3(np_l[val_P(\alpha)])[1,\ldots,b_l[val_P(\alpha)]] ::= np_l[val_P(\alpha)],$$

$$r_3(np_l[val_P(\alpha)])[b_l[val_P(\alpha)]+j] ::= n_{i_j},\ j \in \{1,\ldots,m\},$$

$$r_3(b_l[val_P(\alpha)]) = b_l[val_P(\alpha)]+|N_{np_l}|,$$

$$r_3(f_l[val_P(\alpha)]) ::= f_l[val_P(\alpha)]-|N_{np_l}|,$$

wobei die Werte np_{i_j} Werte von Nachrichtenvariablen sind und die folgenden Beziehungen gelten:

$$N_{np_l} ::= \{n_{i_1},\ldots,n_{i_m}\} = \bigcup_{\alpha[j] \in M_k \;\&\; E(\alpha[j],v)} v[val_P(\alpha_j)],$$

wobei $E(\alpha[j],v) ::= \exists v \in V_{P_{bz(\alpha[i])}}: [\alpha[j] \equiv receive_{RANDOM}(np_l, v)]$.
Die Ermittlung der Werte n_{i_j} kann dadurch unterstützt werden, in dem zu jeder Nachrichtenvariablen, selbst wenn es sich um eine lokale Variable handelt, eine Variablenhistorie geführt wird.
Für die restlichen Variablen $v \in V_{P_{bz(\alpha[i])}}$, es handelt sich nur um lokal definierte

Variablen, und Variablen $v \in V_P - V_{P_{bs(\alpha[i])}}$ kann $\tau_3(v[val_P(\alpha)])$ ausgehend von $val_P(\alpha)$ wie folgt rekonstruiert werden

$$\tau_3(v[val_P(\alpha)]) ::= \begin{cases} v[val_P(\alpha_k)] & \\ & falls \quad v \in V_{P_{bs(\alpha[i])}} \ \& \ l(v) \\ v[val_P(\alpha)] & \\ & \qquad\quad v \in V_P - V_{P_{bs(\alpha[i])}} \end{cases}$$

Die wesentliche Voraussetzung, die den Beweis des Satzes 3.4.4/1 ermöglicht, ist die Annahme P sei ein *RANDOM*-Nachrichtensystem. Dadurch können die Zustände $\tau_3(np_i[val_P(\alpha)])$ und $np_i[val_P(\tau_3(\alpha))]$ als gleichwertig angesehen werden, obwohl die rekonstruierte Reihenfolge der Nachrichten im Puffer np_i nicht zu derjenigen im Zustand $val_P(\tau_3(\alpha))$ identisch sein muß, denn unter der Voraussetzung *RANDOM*-Nachrichtensystem kann die Reihenfolge der Nachrichten in einem Nachrichtenpuffer als vertauschbar (kommutativ) angesehen werden. ■

Analog zu Satz 3.4.3/1 läßt sich auch zu Satz 3.4.4/1 ein Korollar ableiten, das die Aussage von Satz 3.4.4/1 verallgemeinert.

Korollar 3.4.4/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Für P gelte $n(P)$=**true**, d.h. in aktiven Berechnungen α von P sollen nur kommunikationsbedingte Abhängigkeiten bestehen. Die Entnahmestrategie aus Nachrichtenpuffern von P sei *RANDOM*.
α sei eine aktive Berechnung von P, $\alpha[i]$ eine fehlerhafte Aktion, die zurückgesetzt werden muß, und α besitze die im Satz 3.4.4/1 skizzierten Eigenschaften (1) bis (3). Dann ist die in Satz 3.4.4/1 beschriebene Funktion τ_3 Recoveryfunktion zu α.

Beweis: Der Beweis zu Korollar 3.4.4/1 kann analog zum Beweis des Korollars 3.4.3/1 geführt werden und bleibt deshalb dem Leser überlassen. ■

Der Effizienzunterschied (gemessen in der Zahl zurückzusetzender Aktionen) zwischen den Recoveryfunktionen τ_1 und τ_3 läßt sich in einer für τ_3 geeigneten Fehlersituation anhand des folgenden Beispiels eines Erzeuger-/Verbraucher-Nachrichtensystems verdeutlichen:

Beispiel 3.4.4/1: Das in Bild 3.4.4/1 skizzierte Petri-Netz stellt ein Prozeßsystem $P=(A,D,w)$ dar, in dem die Folge

$$\alpha ::= a_{2,1}a_{2,2}(a_{1,1}a_{1,2}a_{1,3})^* a_{1,1}a_{2,3}$$

aktiv ist. Wird die Aktion $a_{2,3}$ in α als fehlerhaft erkannt, so ermitteln die

Recoveryfunktionen r_1 bzw. r_3 die folgenden Wiederaufsetzpunkte:

$r_1(val_P(\alpha))=val_P(a_{2,1})$,
$r_3(val_P(\alpha))=val_P(a_{2,1}(a_{1,1}a_{1,2}a_{1,3})^*a_{1,1})$

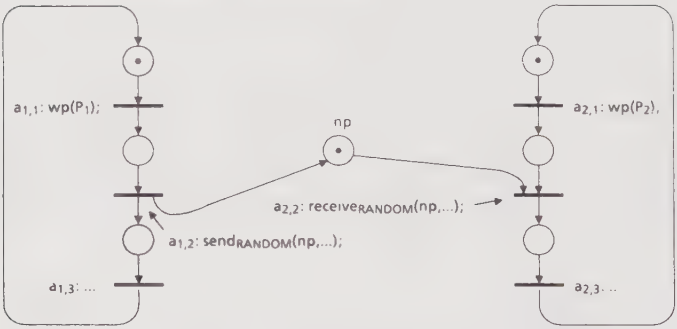


Bild 3.4.4/1: Typische Kommunikationsbeziehung zwischen zwei Prozessen

Das Ziel dieses Abschnittes war es eine Recoveryfunktion r_3 vorzustellen, welche den Fehlerbehandlungsaufwand im Falle der *Rückgabe empfangener Nachrichten* im Vergleich zur allgemeinen Recoveryfunktion r_1 zu reduzieren gestattet. Der Recoveryaufwand läßt sich natürlich auch im Falle der *Rücknahme gesendeter Nachrichten*, abhängig von der Nachrichteneinfüge-/Nachrichtenentnahme-Strategie (FIFO, LIFO oder RANDOM), durch geeignete Fehlerbehandlungsmaßnahmen gegenüber r_1 verringern. Die Vertiefung dieses Punktes bleibt dem Leser überlassen.
Ein weiterer wichtiger Aspekt, der hier erwähnt werden soll, bezieht sich auf die Definition des Begriffes Recoveryfunktion (vgl. Definition 3.4.1/1) selbst. Als wesentliche Eigenschaft wurde hier verlangt, daß für eine Recoveryfunktion r gelten soll.

$$(*) \quad r(val_P(\alpha))=val_P(r(\alpha)).$$

Diese Forderung könnte prinzipiell auch in der folgenden Weise abgeschwächt werden, in dem statt (*) gefordert würde:

$$(**) \quad r(val_P(\alpha))=z,$$

wobei für z in diesem Falle entweder

$$(**.1) \quad \beta \text{ aktiv in } P=(A,D,z) \iff \beta \text{ aktiv in } P=(A,D,val_P(r(\alpha)))$$

oder

(**.2) β aktiv in $P=(A,D,z)$ => β aktiv in $P=(A,D,val_P(r(\alpha)))$

zusätzlich gelten muβte

Strenggenommen muβ bereits beim Beweis von Satz 3.4.4/1 auf die Definition (**) zurückgegriffen werden, denn, wie in der Beweisskizze ausgeführt wurde, muβ die Reihenfolge der Nachrichten in den Nachrichtenpuffern in den Zuständen $r_3(val_P(\alpha))$ und $val_P(r_3(\alpha))$ nicht identisch sein. In jedem Falle besitzt der Zustand $r_3(val_P(\alpha))$ jedoch die Eigenschaft (**.1).
Aber auch die alleinige Existenz von Eigenschaft (**.2) könnte für eine Recoveryfunktion, die der Beziehung (**) genügt, ausreichend sein, denn in der Praxis ist man häufig bereit, nach dem Auftreten eines Fehlers den Betrieb eines Rechensystems gegebenenfalls mit eingeschränkter Leistungsfähigkeit fortzusetzen, falls eine Mindestfunktionalität aufrechterhalten werden kann. Der Punkt (**.2) zielt in diese Richtung, müβte bei Zugrundelegung aber durch weitere Forderungen ergänzt werden, denn ein Zustand z, in welchem keine Berechnung α aktiv ist, erfüllt ebenfalls trivialerweise die Eigenschaft (**.2), entspricht aber einem Systemstillstand, den man in der Regel mit möglichst geringem Aufwand beseitigen will.

Es ist offensichtlich, daβ der Fehlerbehandlungsaufwand (Rücksetzaufwand) um so geringer gehalten werden kann, je weniger streng die Anforderungen eine Recoveryfunktion betreffend sind. Die Anforderungen an die Eigenschaften einer Recoveryfunktion beeinflussen damit auch die Möglichkeiten zur Abschätzung des maximal zu erwartenden Recoveryaufwandes im Fehlerfall und damit Aussagen darüber, ob ein Dominoeffekt auftreten kann oder nicht, was ein zentrales Anliegen dieser Arbeit ist, das im nächsten Kapitel aufgegriffen wird.

Abschlieβend sei im Zusammenhang mit der Definition von Recoveryfunktion r_3 noch auf zwei in [AL81] skizzierten Ideen verwiesen. Zum einen wird vorgeschlagen, bei Zurücksetzung eines Senders die Zurücksetzung der Empfänger seiner gesendeten Nachrichten von der Bedeutung der erhaltenen Information für diese abhängig zu machen. Zum anderen wird die Möglichkeit diskutiert, jede gesendete Information durch ihre Empfänger verifizieren zu lassen, bei erfolgreicher Verifikation kann der Sender dieser Information dann nicht für den Fehler eines Empfängers verantwortlich gemacht werden, d.h. er muβ deswegen prinzipiell nicht zurückgesetzt werden.
Der Recoveryfunktions-Begriff wurde auch in diesen Fällen eine gute Basis für die präzise Formulierung beider Ideen in Form einer Fehlerbehandlungs-Strategie bieten.

4. Charakterisierung des Dominoeffektes

Im vierten Abschnitt des vorangehenden Kapitels wurde gezeigt, daß die Größe des Fehlerbehandlungsaufwandes, gemessen in der Zahl zurückzusetzender Aktionen und deshalb als Rücksetzaufwand zu bezeichnen, von der Wahl der zugrundegelegten Recoveryfunktion abhängt. In den nun folgenden Abschnitten dieses Kapitels wird, unter Zugrundelegung der in bezug auf das in Abschnitt 2.2 vorgestellte Fehlerbehandlungs-Modell allgemeingültigen Recoveryfunktion r_1, der Frage nachgegangen, welche Bedingungen erfüllt sein müssen, um den im Fehlerfall entstehenden Rücksetzaufwand zu einer beliebigen Berechnung α eines gegebenen, aber beliebigen Prozeßsystems P durch eine von α unabhängige Konstante $n \in \mathbf{N}$ a priori als beschränkbar bezeichnen und auf diese Weise den Dominoeffekt, eine Präzisierung dieses Begriffes erfolgt in Definition 4.1/1, vermeiden zu können. Dabei gehen die Überlegungen zunächst von der Annahme der Nichtberücksichtigung potentieller Fehlerverursacher, d.h. dem Problem der Rücksetzausbreitung, aus. Eine Verallgemeinerung auf den Fall der Berücksichtigung potentieller Fehlerverursacher, d.h. des Problems der Fehlerausbreitung, erfolgt anschließend.

Die Charakterisierung des Dominoeffektes in diesem Kapitel wird den Nachweis der Dominoeffektfreiheit verschiedener Prozeßsystemklassen in Kapitel fünf ermöglichen und damit die Grundlage für Überlegungen zur Konstruktion dominoeffektfreier Systeme bereitstellen.
Ausgehend von der Unterscheidung zweier Teilprobleme wird der Dominoeffekt sowohl im Zusammenhang mit dem Problem der Begrenzung der Rücksetzausbreitung als auch dem Problem der Begrenzung der Fehlerausbreitung untersucht, und es werden Bedingungen für Dominoeffektfreiheit abgeleitet.

4.1. Unterscheidung zweier Teilprobleme

Wie bereits angedeutet, wird in dieser Arbeit der in einer beliebigen Fehlersituation innerhalb einer Berechnung α eines Prozeßsystems P zu erbringende Fehlerbehandlungsaufwand, gemessen in der Zahl zurückzusetzender Aktionen, als ein Kriterium für das *Auftreten/Nichtauftreten eines Dominoeffektes* herangezogen. Präzise formuliert werden die Berechnungen α eines Prozeßsystems $P=(A,D,w)$ genau dann als *dominoeffektfrei* bezeichnet, wenn P der folgenden Definition genügt:

Definition 4.1/1: Ein Prozeßsystem $P=(A,D,w)$ wird als *dominoeffektfrei* bezeichnet genau dann, wenn gilt:

$$\exists n \in \mathbf{N}: [\forall \alpha: [\alpha \text{ aktive Berechnungsfolge von } P \Rightarrow |Z_\alpha(\bigcup_{\alpha[i] \in V(\alpha) \,\&\, f_\alpha(\alpha[i])} F_{w(\alpha)}(\alpha[i]))| \leq n]].$$

Falls es zu P kein derartiges $n \in \mathbb{N}$ gibt, so sagt man, daß in P im Recoveryfall der *Dominoeffekt* auftreten kann.

Bemerkung: Die Präzisierung des Begriffes Dominoeffekt ist in dieser Arbeit so gewählt, daß andere, in der Literatur, wie etwa in [Rus76], [Rus77], [Rus80] und [Kan81], intuitiv mit dieser Bezeichnung verbundenen Vorstellungen, sich als Spezialfälle dieser Präzisierung modellieren lassen.
K. Kant [Kan 81] sieht im Dominoeffekt, ohne dies genau zu präzisieren, das *Problem der unbeschränkbaren zyklischen Zurücksetzung* (dieses Problem wird in dieser Arbeit im Rahmen des Abschnittes 4.2 beschrieben und als Teilaspekt vertieft), und D.L. Russell betrachtet die *Unbeschränkbarkeit der vermeidbaren Rücksetzung* (eine genaue Definition und Untersuchung dieses Punktes als einem Bestandteil, der zum Dominoeffekt führen kann, erfolgt ebenfalls in Abschnitt 4.2) als den Kern des Problems Dominoeffekt, wobei er seine Vorstellungen als einer der wenigen Autoren, welche sich mit diesem Problemkreis beschäftigen, vergleichsweise genau erläutert und formal skizziert.
Dazu ist zu ergänzen, daß, wie in [Rus80] angemerkt, die Beschränkbarkeit der *vermeidbaren Zurücksetzung* auch die Beschränkbarkeit der *zyklischen Zurücksetzung* zur Folge hat, jedoch nicht zwangsläufig umgekehrt. Im Sinne von Definition 4.1/1 ist die Beschränkbarkeit beider Phänomene Voraussetzung für das Nichtauftreten des Dominoeffektes.

Ein Algorithmus, der im Falle der Zugrundelegung von Recoveryfunktion r_1 die zurückzusetzenden Aktionen einer Berechnung α ermittelt, kann (logisch gesehen) in den folgenden vier Phasen ablaufen:

Phase 1: Erkennung aller aufgetretenen Fehler und Lokalisierung aller ihrer (nichtverifizierten) potentiellen Verursacher.

Phase 2: Rücksetzen aller potentiellen Fehlerverursacher und der mit diesen eine Recoveryeinheit bildenden Aktionen.

Phase 3: Rücksetzen aller auf den in der vorangehenden *Phase 2* (bzw. *Phase 4*) rückgesetzten Aktionen basierenden Aktionen, soweit sie noch nicht rückgesetzt worden sind.
Ist die Menge der neu rückzusetzenden Aktionen leer, beenden des Algorithmus; ansonsten weiter mit *Phase 4*.

Phase 4: Rücksetzen aller mit den in der vorangehenden *Phase 3* rückgesetzten Aktionen eine Recoveryeinheit bildenden Aktionen, soweit diese noch nicht rückgesetzt worden sind.
Ist die Menge der neu rückzusetzenden Aktionen leer, beenden des Algorithmus; ansonsten weiter mit *Phase 3*.

In der *Phase 1* wird die Menge $\bigcup_{a[i]\in W(a)\ \&\ f_a(a[i])} F_{nv(a)}(a[i])$ bestimmt und mittels der Folge

$$(\textit{Phase 2}, \textit{Phase 3}[[,(\textit{Phase 4}, \textit{Phase 3})^*][, \textit{Phase 4}]])$$

wird von der Menge der (nichtverifizierten) potentiellen Fehlerverursacher ausgehend die Menge

$$(*)\quad Z_a\Big(\bigcup_{a[i]\in W(a)\ \&\ f_a(a[i])} F_{nv(a)}(a[i])\Big)$$

errechnet.
Mit einer ähnlichen Beweistechnik, wie sie im Beweis zu Hilfssatz 3.4.3/1 zum Einsatz kam, läßt sich zeigen, daß die Menge (*) mit der wie folgt definierten Menge übereinstimmt:

$$(**)\quad \bigcup_{a[i]\in W(a)\ \&\ f_a(a[i])} Z_a(F_{nv(a)}(a[i])).$$

Ausgehend von (**) ist die folgende Schlußfolgerung zulässig:

Hilfssatz 4.1/1: Ein Prozeßsystem $P=(A,D,w)$ ist dominoeffektfrei genau dann, wenn es ein $m\in\mathbf{N}$ gibt, so daß für *alle* aktiven Berechnungen α von P gilt:

(1) die Zahl der Aktionen $a[i]\in W(\alpha)$, für welche $f_a(a[i])=$**true** gilt, ist kleiner oder gleich m,

(2) ist $\alpha[i]$ eine fehlerhafte Aktion, so ist $|Z_a(F_{nv(a)}(a[i]))|\leq m$.

Beweis: Legt man die Definition 4.1/1 zugrunde, so errechnet sich unter den Voraussetzungen des Hilfssatzes die Konstante n, die gemäß Definition 4.1/1 zum Nachweis der Dominoeffektfreiheit von P genügt, zu $m*m$. ■

Da sich, wiederum mit der gleichen Beweistechnik wie im Beweis zu Hilfssatz 3.4.3/1, die Menge $Z_a(F_{nv(a)}(a[i]))$ als Vereinigung von Mengen $Z_a(a[j])$, wobei $a[j]\in F_{nv(a)}(a[i])$ gilt, darstellen läßt:

$$(***)\quad Z_a(F_{nv(a)}(a[i])) = \bigcup_{a[j]\in F_{nv(a)}(a[i])} Z_a(a[j]),$$

ist des weiteren der folgende Schluß zulässig

Hilfssatz 4.1/2: In der Situation des Hilfssatzes 4.1/1 ist die Beziehung (2), $Z_\alpha(F_{nv(\alpha)}(\alpha[i])) \leq m$, genau dann gültig, wenn es zwei Konstanten $m_1,m_2 \in \mathbf{N}$ gibt, so daß für alle Berechnungen α von P gilt:

(1) ist $\alpha[i]$ eine fehlerhafte Aktion, so ist $|F_{nv(\alpha)}(\alpha[i])| \leq m_1$ und

(2) für jedes $\alpha[j] \in F_{nv(\alpha)}(\alpha[i])$, $\alpha[i]$ fehlerhaft, gilt: $|Z_\alpha(\alpha[j])| \leq m_2$.

Beweis: Unter den Voraussetzungen des Hilfssatzes und der Zugrundelegung der Beziehung (•••) läßt sich die Konstante m in Voraussetzung (2) des Hilfssatzes 4.1/1 in der skizzierten Situation zu $m_1 * m_2$ bestimmen. ■

Die Aussagen der Hilfssätze 4.1/1 und 2 lassen sich schließlich im folgenden Korollar zusammenfassen:

Korollar 4.1/1: Ein Prozeßsystem $P=(A,D,w)$ ist dominoeffektfrei genau dann, wenn es drei Konstanten $m_1,m_2,m_3 \in \mathbf{N}$ gibt, so daß für *alle* aktiven Berechnungen α von P gilt:

(1) die Zahl der Aktionen $\alpha[i] \in W(\alpha)$, für welche $f_\alpha(\alpha[i])$=**true** gilt, ist kleiner oder gleich m_1,

(2) ist $\alpha[i]$ eine fehlerhafte Aktion, so ist $|F_{nv(\alpha)}(\alpha[i])| \leq m_2$ und

(3) für jedes $\alpha[j] \in F_{nv(\alpha)}(\alpha[i])$, $\alpha[i]$ fehlerhaft, gilt: $|Z_\alpha(\alpha[j])| \leq m_3$.

Beweis: Folgt unmittelbar mit den Aussagen der Hilfssätze 4.1/1 und 2. ■

Mit dem Ergebnis von Korollar 4.1/1 (bzw. Hilfssatz 4.1/1) ist eine der Bedingungen für die Dominoeffektfreiheit eines Prozeßsystems $P=(A,D,w)$ in der Existenz einer Konstanten $m \in \mathbf{N}$ ($m_1 \in \mathbf{N}$), mittels welcher sich die Zahl der fehlerhaften Aktionen jeder beliebigen aktiven Berechnung α von P beschränken läßt, gefunden. Im weiteren gilt es nun die Punkte (Teilprobleme) (2) und (3) von Korollar 4.1/1 weiter zu verfeinern und durch transparentere Bedingungen zu ersetzen.
Bei Teilproblem (2) geht es speziell darum, Bedingungen zu finden, unter welchen die Zahl der potentiellen Fehlerverursacher eines beliebigen Fehlers innerhalb jeder aktiven Berechnung α eines Prozeßsystems P a priori begrenzt werden kann; Lösungen für diesen Bereich zu finden ist ein Anliegen von Abschnitt 4.3.
Zu Teilproblem (3) sind wiederum Kriterien abzuleiten, mittels derer die

Rücksetzausbreitung bei der Korrektur der Änderungen einer fehlerhaften Aktion im voraus auf eine maximale Anzahl von Aktionen eingeschränkt werden kann. Nähere Untersuchungen in diese Richtung sind der Gegenstand des nächsten Abschnitts.

4.2. Dominoeffekt und Rücksetzausbreitung

Wie im vorangehenden Abschnitt 4.1 skizziert, besteht eine der Voraussetzungen für das Nichtauftreten des Dominoeffektes in der Begrenzbarkeit der Rücksetzausbreitung, d.h. in der Existenz einer Konstanten $n \in \mathbb{N}$, so daß für jede beliebige fehlerhafte Aktion $\alpha[i]$ einer beliebigen Berechnung α und jedes $\alpha[j] \in F_{rw(\alpha)}(\alpha[i])$ eines Prozeßsystems $P=(A,D,w)$ gilt: $|Z_\alpha(\alpha[j])| \leq n$.
Das Ableiten elementarer Bedingungen, die das Erfülltsein dieser Voraussetzung gewährleisten, ist das Ziel dieses Abschnittes. Es wird in folgenden Schritten verfolgt: Zunächst wird die Menge $Z_\alpha(\alpha[j])$ in die *unvermeidbar* und *vermeidbar rücksetzenden Aktionen* unterteilt, danach werden einige Betrachtungen durchgeführt und Hilfssätze bewiesen, welche schließlich Bedingungen zur Begrenzung der Rücksetzausbreitung abzuleiten gestatten.

4.2.1. Unvermeidbare und vermeidbare Rücksetzausbreitung

Bei der Rücksetzausbreitung lassen sich zwei Komponenten unterscheiden, welche die Menge $Z_\alpha(\alpha[j])$ in unvermeidbar und vermeidbar rückzusetzende Aktionen zu trennen gestatten. Genauer gesagt handelt es sich dabei um zwei disjunkte Teilmengen $U_{Z_\alpha}(\alpha[j])$ bzw. $V_{Z_\alpha}(\alpha[j])$.
Der Bestandteil der *unvermeidbaren Rücksetzausbreitung* ist dadurch charakterisiert, daß durch ihn ausschließlich fehlerhafte Aktionen und die auf diesen basierenden Aktionen zurückgesetzt werden, wohingegen mittels der *vermeidbaren Rücksetzausbreitung* diejenigen Aktionen zurückgesetzt werden, die zwar berechnungstechnisch im Sinne der Relation B_α^* von keiner fehlerhaften Aktion abhängig sind, jedoch recoverystrategisch mit einer solchen Aktion im Sinne der Relation E_α verbunden (durch den gleichen Wiederaufsetzpunkt gesichert) sind. Damit lassen sich die Teilmengen $U_{Z_\alpha}(\alpha[j])$ bzw. $V_{Z_\alpha}(\alpha[j])$ wie folgt definieren:

Definition 4.2.1/1: Sei α ein aktiver Ablauf eines Prozeßsystems $P=(A,D,w)$ und $\alpha[i]$ eine Aktion der Folge α. Dann ist die Menge $U_{Z_\alpha}(\alpha[i])$ der Aktionen, deren Rücksetzung beim notwendig werdenden Zurücksetzen der (fehlerhaften oder durch einen Rechensystemfehler unterbrochenen) Aktion $\alpha[i]$ (zum Wiederaufsetzpunkt $WP_\alpha(\alpha[i])$) bei Nichtberücksichtigung potentieller Fehlerverursacher *unvermeidbar* ist, im Einklang mit der Präzisierung dieses Begriffes in [Rus80], definiert als:

$$U_{Z_\alpha}(\alpha[i]) ::= \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; (\alpha[j] B_\alpha^* \alpha[i] \parallel \alpha[j] E_\alpha \alpha[i])\}.$$

Die Menge $V_{Z_\alpha}(\alpha[i])$ der Aktionen, deren Rücksetzung in der skizzierten Situation bei Nichtberücksichtigung potentieller Fehlerverursacher *vermeidbar* ist, läßt sich wie folgt beschreiben:

$$V_{Z_\alpha}(\alpha[i]) ::= Z_\alpha(\alpha[i]) - U_{Z_\alpha}(\alpha[i]).$$

Bemerkung: Eine Verallgemeinerung von Definition 4.2.1/1 auf den Fall der Berücksichtigung potentieller Fehlerverursacher bzw. mehrerer zurückzusetzenden Aktionen ist in Analogie zur Definition 3.2.4/2 offensichtlich. Auf eine explizite Definition wird daher verzichtet.
Prinzipiell hätte die Menge $U_{Z_\alpha}(\alpha[i])$ auch wie folgt definiert werden können:

$$U_{Z_\alpha}(\alpha[i]) ::= \{\alpha[j] \mid \alpha[j] \in W(\alpha) \;\&\; \alpha[j] B_\alpha^* \alpha[i]\},$$

d.h. nur die Zurücksetzung der auf einer fehlerhaften Aktion basierenden Aktionen hätte als unvermeidbar bezeichnet werden können. In Definition 4.2.1/1 wurde jedoch zusätzlich auch die Zurücksetzung der mit $\alpha[i]$ eine Recoveryeinheit bildenden Aktionen als unvermeidbar eingestuft, da Aktionen einer Recoveryeinheit häufig nicht nur recoverystrategisch, sondern auch logisch eine Einheit bilden, und ein Fehler in der Regel (durch einen Akzeptanztest) nicht genau auf eine einzelne Aktion, sondern nur auf eine Gruppe logisch zusammengehörender Aktionen eingegrenzt werden kann.

Eine unmittelbare Folgerung der vorstehenden Definition ist die Aussage des folgenden Hilfssatzes:

Hilfssatz 4.2.1/1: Sei α eine aktive Berechnung eines Prozeßsystems $P=(A,D,w)$, $\alpha[i] \in W(\alpha)$. Dann gilt:

(1) $Z_\alpha(\alpha[i]) = U_{Z_\alpha}(\alpha[i]) \cup V_{Z_\alpha}(\alpha[i])$ und

(2) $U_{Z_\alpha}(\alpha[i]) \cap V_{Z_\alpha}(\alpha[i]) = \phi$.

Beweis: Folgt unmittelbar aufgrund der Definition der Menge $V_{Z_\alpha}(\alpha[i])$ (vgl. Definition 4.2.1/1). ■

Analog zur Aufspaltung des Problems Dominoeffekt in die zwei Teilprobleme Begrenzung der Fehlerausbreitung und Begrenzung der Rücksetzausbreitung, läßt sich auf der Basis von Definition 4.2.1/1 das Problem der Rücksetzausbreitung entsprechend weiter unterteilen in das Problem der Begrenzung der vermeidbaren und das Problem der Begrenzung der *sowohl* vermeidbaren *als auch* unvermeidbaren Rücksetzausbreitung. Bevor jedoch Lösungen zu den beiden letztgenannten Problemen in den Abschnitten 4.2.3 bzw. 4.2.4 präzisiert werden können, werden im nachfolgenden Abschnitt zunächst einige Vorüberlegungen erläutert und später benötigte Hilfssätze bewiesen.

4.2.2. Betrachtungen zur Begrenzung der Rücksetzausbreitung

Man überlegt sich leicht, daß ein Dominoeffekt immer dann einkalkuliert werden muß, wenn zumindest eine der drei folgenden Bedingungen erfüllt ist:

(1) das Problem der *unbeschränkbaren zyklischen Zurücksetzung* auftritt oder

(2) das Problem der *unbeschränkbar großen Recoveryeinheiten* gegeben ist oder

(3) das Problem der *unbeschränkbar langen Prozeßinaktivphasen* besteht.

Daß die drei mit (1) bis (3) bezeichneten Bedingungen *voneinander unabhängig* sind, zeigen die folgenden Erläuterungen und die Bilder 4.2.2/1 bis 4.2.2/3, anhand derer sie veranschaulicht werden:

zu(1) Das Problem der *unbeschränkbaren zyklischen Zurücksetzung* (1) tritt in einem Prozeßsystem P genau dann auf (vgl. Bild 4.2.2/1), wenn es in P aktive Berechnungsfolgen α und Aktionen $\alpha[i] \in W(\alpha)$ gibt, so daß die Zurücksetzung einer fehlerhaften Aktion $\alpha[i]$ im Zustand $val_P(\alpha)$ die sukzessive Zurücksetzung eines zu P gehörenden Prozesses P_j zu den Sicherungspunkten $WP_\alpha(\alpha[j_1]),\ldots,WP_\alpha(\alpha[j_n])$ erfordert, wobei:

$$t(WP_\alpha(\alpha[i])) > t(WP_\alpha(\alpha[j_1])) > \ldots > t(WP_\alpha(\alpha[j_n])) \;\&\; \forall l \in \{1,\ldots,n\}: \neg wp(\alpha[j_l])$$

gilt, und die Länge n dieser Rücksetzfolge nicht unabhängig von α vorab beschränkt werden kann.

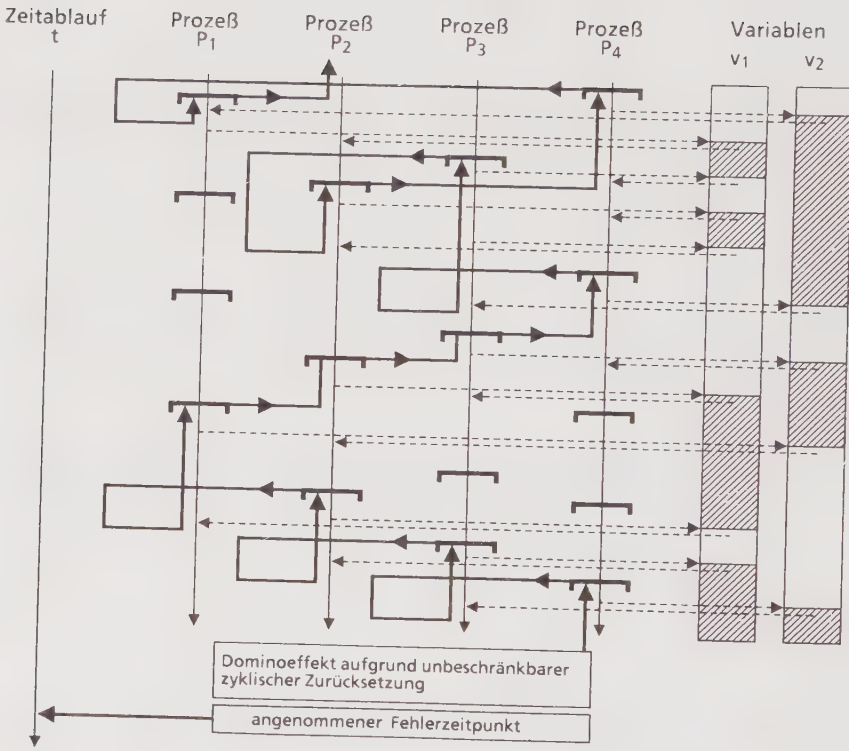
zu(2). Das Problem der *unbeschränkbar großen Recoveryeinheiten* (2) ist in einem Prozeßsystem P genau dann gegeben (vgl. Bild 4.2.2/2), wenn es kein $n \in \mathbf{N}$ gibt, so daß für alle aktiven Berechnungsfolgen α von P und Aktionen $\alpha[i] \in W(\alpha)$ gilt:

$$|\{\, j \mid \alpha[j] \in W(\alpha) \;\&\; \alpha[i] E_\alpha \alpha[j] \}| < n.$$

zu(3) Das Problem der *unbeschränkbar langen Prozeßinaktivphasen* (3) besteht in einem Prozeßsystem $P=(A,D,w)$ genau dann (vgl. Bild 4.2.2/3), wenn gilt:

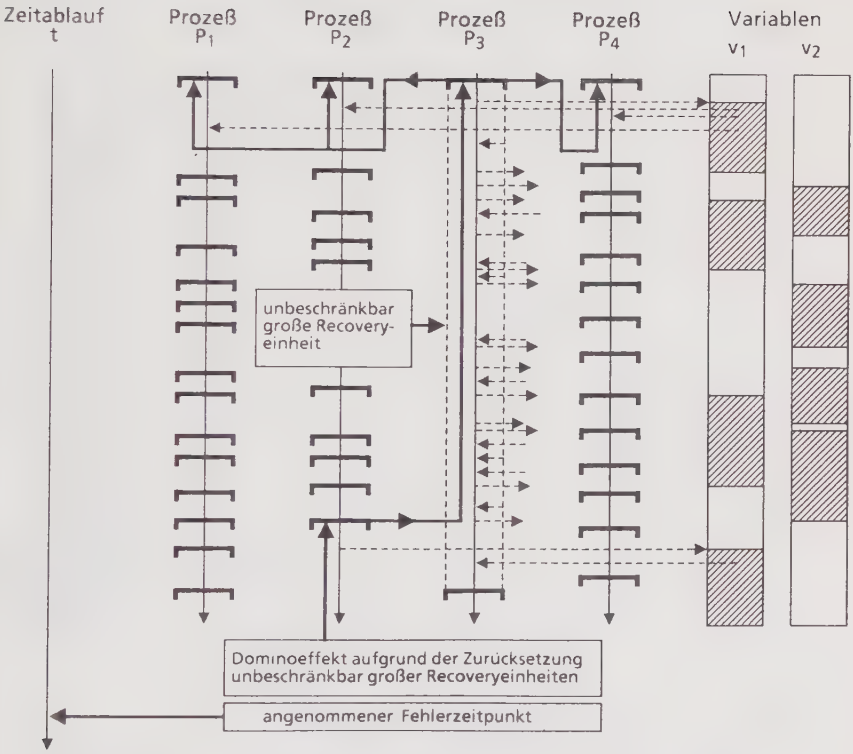
$$\neg \exists n \in \mathbf{N}: [\forall \alpha \in A\ \forall \alpha,\beta: [\alpha \in bef_P(\alpha) \;\&\; \alpha\beta \text{ aktiv} \;\&\; P_{bz(\alpha)} \cap W(\beta) = \phi] \Rightarrow |\beta| < n],$$

d h wenn nicht garantiert werden kann, daß jeder (nichtbeendeter) Prozeß P_i von P mindestens alle n Aktionen einmal aktiv ist.



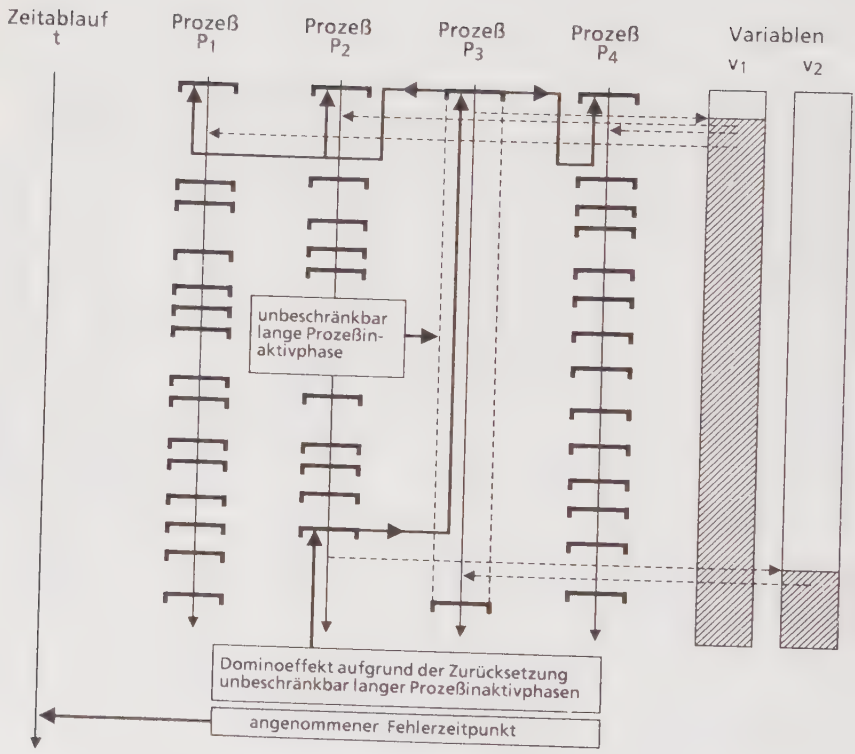
Legende: siehe Bild 1.3/1 und 3;

Bild 4.2.2/1: Veranschaulichung des Problems der unbeschränkbaren zyklischen Zurücksetzung



Legende: siehe Bild 1 3/1 und 3,

Bild 4.2.2/2: Veranschaulichung des Problems der unbeschränkbar großen Recoveryeinheiten



Legende: siehe Bild 1 3/1 und 3;

Bild 4.2.2/3: Veranschaulichung des Problems der unbeschränkbar langen Prozeßinaktivphasen

Die Bilder 4.2.2/1 und 2 veranschaulichen, daß die Punkte (1) bzw. (2) hinreichend sind für das Auftreten des Dominoeffektes. Satz 4.2.2/1 hält dies fest.

Satz 4.2.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Falls zu jedem $n\in\mathbb{N}$ in P eine aktive Berechnung α existiert, in der:

(1) zyklische Zurücksetzung der Länge größer n oder

(2) Recoveryeinheiten größer n

vorkommen, dann kann in P der Dominoeffekt auftreten.

Beweis: Vergleiche die Bilder 4.2.2/1 und 4.2.2/2. ■

Bedingung (3) allein ist nicht hinreichend. Denn wie leicht einzusehen ist, kann ein Prozeß P_i eines Prozeßsystems $P=(A,D,w)$ durchaus unbeschränkbar lange Prozeßinaktivphasen haben, ohne daß deswegen bereits ein Dominoeffekt auftreten muß, vorausgesetzt beispielsweise, daß er von den anderen Prozessen unabhängig ist.

Im nachsten Abschnitt wird gezeigt, daß unter bestimmten Voraussetzungen die Punkte (1) und (2) des Satzes 4.2.2/1, ebenso wie eine modifizierte Fassung der Bedingung (3), nicht nur hinreichende, sondern auch notwendige Bedingungen für das Auftreten des Dominoeffektes darstellen
Um einen prazisen und kompakten Beweis dieses Sachverhaltes zu ermöglichen, werden im folgenden zwei Hilfssätze abgeleitet, die auf die Bedingungen (2) und (3) bezug nehmen.

Bemerkung: Immer dann, wenn im folgenden die Beziehung $\alpha[j](B_\alpha\cup E_\alpha)^*\alpha[i]$ zwischen zwei Aktionen einer Berechnungsfolge α als gegeben angenommen und unter bezug auf Hilfssatz 3.2.4/5 auf die Existenz von Aktionen $\alpha[j_1],\ldots,\alpha[j_m]$ mit der Eigenschaft:

$$\alpha[j]=\alpha[j_m](B_\alpha\cup E_\alpha)\alpha[j_{m-1}]\ldots\alpha[j_2](B_\alpha\cup E_\alpha)\alpha[j_1]=\alpha[i]$$

geschlossen wird, bedeutet dies selbstverständlich, daß - auch ohne besondere Erwahnung - die Aktionen $\alpha[j_l]$, $1<l<m$, genauso wie die Aktionen $\alpha[j]$ und $\alpha[i]$ der Menge $W(\alpha)$ angehoren, denn $(B_\alpha\cup E_\alpha)\subseteq W(\alpha)\times W(\alpha)$.
$t(WP(\ ))$ bezeichnet in den folgenden Hilfssätzen den Zeitpunkt, zu welchem das Anlegen des Wiederaufsetzpunktes $WP(\ldots)$ erfolgreich abgeschlossen wurde.

Definition 4.2.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem, α aktiv in P.

$$Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))$$

sei die Menge der Aktionen, die im Zustand $val_P(\alpha)$ zurückgesetzt werden müssen. Dann ist die Menge

$$Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t_1,t_2]),$$

die *Restriktion von* $Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))$ *auf Aktionen des Zeitintervalls* $[t_1,t_2]$, $t_1\le t_2$, wie folgt definiert:

$$Z_\alpha(\bigcup_{\alpha[i]\subset W(\alpha)\&f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t_1,t_2]) ::=$$

$$\{\alpha[j] \mid \alpha[j]\in Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i])) \;\&\; t_1\le t(\alpha[j])\le t_2\}$$

$[t_1,t_2[$, $t_1\le t_2$, wird im folgenden abkürzend für das Intervall $[t_1,t_2]$, jedoch ohne den Zeitpunkt t_2, geschrieben.
$]t_1,t_2[$ steht stellvertretend für das Intervall $[t_1,t_2]$, jedoch ohne t_1 und t_2.

Hilfssatz 4.2.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem, αx sei, $x\in A$, eine aktive Berechnungsfolge von P. Des weiteren gelte

$$\alpha[j]=\alpha[j_m]\langle E_{\alpha x}\cup B_{\alpha x}\rangle\alpha[j_{m-1}] \quad \alpha[j_2]\langle E_{\alpha x}\cup B_{\alpha x}\rangle\alpha[j_1]=x,$$

und die folgenden beiden Bedingungen (1) und (2) seien erfüllt:

(1) $\quad \exists n_1\in\mathbb{N}-\{1\}\forall\beta[k]\in W(\beta)\;\; [\beta \text{ aktiv in } P \Rightarrow |\{i,\beta[i]\in W(\beta)\&\beta[k]E_\beta\beta[i]\}<n_1]$.

(2) $\quad \exists n_2\in\mathbb{N}\forall b\in A\forall\beta_1,\beta_2: [b\in be f_P(\beta_1) \;\&\; \beta_1\beta_2 \text{ aktiv in } P \;\&\; P_{bx(b)}\cap W(\beta_2)=\phi \Rightarrow |\beta_2|<n_2]$.

Unter diesen Voraussetzungen läßt sich für $1<l\le m$ zeigen (der Index "αx" wird im folgenden an den Stellen, wo der Bezug eindeutig ist, weggelassen, um die Darstellung zu komprimieren und die Lesbarkeit zu erhohen. Z, E, B, f steht im folgenden für $Z_{\alpha x}$, $E_{\alpha x}$, $B_{\alpha x}$, $f_{\alpha x}$ und WP stellvertretend für $WP_{\alpha x}$):

(a) $(t(\alpha[j_l])\ge t(\alpha[j_{l-1}])) \Rightarrow$
$(Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_{l-1}]),t(\alpha[j_l])]) \subseteq$
$Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_{l-1}]),t(x)]))$;

(b) $(t(\alpha'j_i]) \le t(\alpha'j_{i-1}])) \Rightarrow$

$$\left| Z\Big(\bigcup_{\alpha[s]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(\alpha'j_i]), t(\alpha'j_{i-1}])]\Big)\right| \le$$

$$(n_1-1)+(n_1-2)*(n_2-1) < n_1 * n_2 \;\&\; \alpha'j_i]E\alpha'j_{i-1}]).$$

(c) $(t(\alpha'j_i]) \le t(\alpha'j_{i-1}])) \Rightarrow$

$$\left| Z\Big(\bigcup_{\alpha[s]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(\alpha'j_i]), t(\alpha'j_{i-1}])]\Big)\right| \le$$

$$Z\Big(\bigcup_{\alpha's]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(WP(\alpha'j_{i-1})), t(\alpha'j_{i-1})])\Big| \le$$

$$(n_1-1)*n_2 < n_1*n_2).$$

(d) $Z\Big(\bigcup_{\alpha's]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(\alpha'j_i]), t(\alpha'j_{i-1}])]\Big)\Big| \le$

$$Z\Big(\bigcup_{\alpha's]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(WP(\alpha'j_{i-1})), t(\alpha'j_{i-1})])\Big| \le$$

$$(n_1-1)*n_2 < n_1*n_2.$$

(e) $Z\Big(\bigcup_{\alpha's]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(WP(\alpha'j_i])), t(\alpha'j_i])]\Big) < (n_1-1)*n_2 < n_1*n_2$

Beweis Im folgenden werden die Behauptungen (a) bis (d) nacheinander bewiesen.

zu(a) Da x die aktuellste Aktion der Folge αx ist, folgt unter der Annahme $t(\alpha'j_i]) < t(\alpha'j_{i-1}])$

$$[t(\alpha'j_{i-1}]), t(\alpha'j_i])] \subseteq [t(\alpha'j_{i-1}]), t(x)]$$

und damit unter Einbeziehung von Definition 4.2.2/1 auch

$$Z\Big(\bigcup_{\alpha[s]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(\alpha'j_{i-1}]), t(\alpha'j_i])]\Big) \subseteq$$

$$Z\Big(\bigcup_{\alpha's]\in W(\alpha x)\,\&\,f(\alpha's])} F_{nv(\alpha x)}(\alpha'i]) \cdot [t(\alpha'j_{i-1}]), t(x)]\Big).$$

zu(b) Da $t(\alpha'j_i]) \le t(\alpha'j_{i-1}])$ und nach Annahme $\alpha'j_i](E\cup B)\alpha'j_{i-1}]$ gilt, muß $\alpha'j_i]E\alpha'j_{i-1}]$ erfüllt sein.
Sei $\alpha'j_i]B\alpha'j_{i-1}]$. Unter der Voraussetzung (1) können im Prozeß P_1 im Zeitintervall $[t(\alpha'j_i]), t(\alpha'j_{i-1}])]$ maximal n_1-1 Aktionen und unter Voraussetzung (2) in den restlichen Prozessen von P maximal $(n_1-2)*(n_2-1)$ Aktionen ausgeführt worden sein. Insgesamt können im Intervall $[t(\alpha'j_i]), t(\alpha'j_{i-1}])]$ nicht mehr als $(n_1-1)+(n_1-2)*(n_2-1) < n_1*n_2$ Aktionen ausgeführt worden sein.

zu(c) Da nach Annahme $t(\alpha'j_i]) \le t(\alpha'j_{i-1}])$, ist, wie unter (b) gezeigt, $\alpha'j_i]E\alpha'j_{i-1}]$ oder von $\alpha'j_i]B\alpha'j_{i-1}]$ kann weiter auf $WP(\alpha'j_{i-1}]) = WP(\alpha'j_i])$ und $t(WP(\alpha'j_{i-1}])) < t(\alpha'j_i])$ geschlossen werden.

Damit gilt $[t(\alpha[j_l]),t(\alpha[j_{l-1}])] \subseteq [t(WP(\alpha[j_{l-1}])),t(\alpha[j_{l-1}])]$ und unter Zugrundelegung der Definition 4.2.2/1 folgt Behauptung (c), wenn man berücksichtigt, daß die den Wiederaufsetzpunkt $WP(\alpha[j_{l-1}])$ erzeugende Aktion die aktuellste vor $t(\alpha[j_{l-1}])$ ausgeführte Aktion, die nicht zur Recoveryeinheit von $\alpha[j_{l-1}]$, jedoch zum gleichen Prozeß gehört, ist, und aus diesem Grunde zwischen der den Wiederaufsetzpunkt $WP(\alpha[j_{l-1}])$ erzeugenden Aktion und der ältesten Aktion, die zur Recoveryeinheit von $\alpha[j_{l-1}]$ gehört, n_2-1 Aktionen in den anderen Prozessen ausgeführt worden sein könnten.

zu(d): Aufgrund von (c) genügt es zu zeigen, daß gilt:

$$(*)\ |Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_{l-1}])),t(\alpha[j_{l-1}])])| \leq (n_1-1)*n_2$$

und

$$(**)\ |Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_l]),t(\alpha[j_{l-1}])])| = 0,$$

falls $t(\alpha[j_l])>t(\alpha[j_{l-1}])$

(*) folgt mit Punkt (c), denn der Nachweis von

$$|Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_{l-1}])),t(\alpha[j_{l-1}])])| \leq (n_1-1)*n_2 < n_1*n_2$$

ist unter "zu(c):" ohne Rückgriff auf die Beziehung $t(\alpha[j_l])\leq t(\alpha[j_{l-1}])$ abgeleitet worden, und (**) folgt mit Definition 4.2.2/1, da es keine Aktion $\alpha[k]\in W(\alpha x)$ mit der Eigenschaft $t(\alpha[j_l])\leq t(\alpha[k])\leq t(\alpha[j_{l-1}])<t(\alpha[j_l])$ geben kann.

zu(e): Innerhalb des Intervalls $[t(WP(\alpha[j_l])),t(\alpha[j_l])]$ können, unter der Annahme (1), im Prozeß $P_{bz(\alpha[j_l])}$ maximal n_1-1 Aktionen und in den restlichen Prozessen maximal $(n_1-1)*(n_2-1)$ Aktionen ausgeführt worden sein (vgl. die Beweisführung unter "zu(c) "); insgesamt also $(n_1-1)*n_2$ Aktionen. Damit folgt die Behauptung (e). ■

Hilfssatz 4.2.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem, αx, $x\in A$, aktiv in P, und alle Voraussetzungen des Hilfssatzes 4.2.2/1 seien erfüllt. Unter diesen Bedingungen gilt mit $l\in\{2,\ldots,m\}$, wobei der Index "αx" dort, wo dies die Lesbarkeit erleichtert und nicht zu Mißverständnissen führt, wie bereits im vorangehenden Beweis praktiziert, weggelassen wird:

(a) $Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_l]),t(\alpha[j_{l-1}])]) \cup$
$Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_{l-1}]),t(x)]) \subseteq$
$Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_{l-1}])),t(x)]) \subseteq$

$$Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big) \subset W(\alpha x);$$

(b) $$Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_l]),t(\alpha[j_{l-1}])]\Big) -$$
$$Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_{l-1}]),t(x)]\Big)| \leq$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_{l-1}])),t(\alpha[j_{l-1}])]\Big)| < n_1 * n_2;$$

(c) $$|\bigcup_{i=2}^{m}(Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_l]),t(\alpha[j_{l-1}])]\Big))_l \leq$$
$$|\bigcup_{i=2}^{m+1}(Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_{l-1}])),t(\alpha[j_{l-1}])]\Big))|;$$

(d) $$[|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [\min_{i\in\{1,\ldots,l\}} t(WP(\alpha[j_i])),t(x)]\Big)| <$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_k])),t(x)]\Big)|] =>$$
$$\forall i\in\{1,\ldots,l\}\ [t(WP(\alpha[j_i]))>t(WP(\alpha[j_k]))\ \&\ WP(\alpha[j_i])\neq WP(\alpha[j_k])];$$

(e) $$k := \min\{i \mid t(WP(\alpha[j_i]))<t(WP(\alpha[j_l]))\ \&\ l<i\leq m\} =>$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_k])),t(x)]\Big)| -$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_l])),t(x)]\Big)| < n_1 * n_2$$

Beweis: Nachfolgend werden die Behauptungen (a) bis (e) bewiesen:

zu(a): Aufgrund von Definition 4.2.2/1 reicht es aus nachzuweisen:

$$[t(\alpha[j_l]),t(\alpha[j_{l-1}])]\cup[t(\alpha[j_{l-1}]),t(x)] \subseteq [t(WP(\alpha[j_{l-1}])),t(x)]$$

Dazu wiederum genügt es zu zeigen:

$$t(\alpha[j_l])\in[t(WP(\alpha[j_{l-1}])),t(x)]$$

$t(x)\geq t(\alpha[j_l])$ ist erfüllt, da x die aktuellste Aktion der Folge αx ist und nach Voraussetzung $\alpha[j_l]\in W(\alpha x)$ gilt.
$t(WP(\alpha[j_{l-1}]))\leq t(\alpha[j_l])$ folgt damit, daß nach Voraussetzung $\alpha[j_l]B\alpha[j_{l-1}]$, d.h.:

$$t(\alpha[j_l])\geq t(\alpha[j_{l-1}])\geq t(WP(\alpha[j_{l-1}])),$$

oder $\alpha[j_l]E\alpha[j_{l-1}]$, d h

$$t(WP(\alpha[j_{l-1}]))=t(WP(\alpha[j_l]))<t(\alpha[j_l]).$$

gelten muß.

zu(b): Folgt unmittelbar mit Hilfssatz 4.2.2/1-(d).

zu(c): Wie in Hilfssatz 4.2.2/1 unter (d) gezeigt, gilt:

$$Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_l]),t(\alpha[j_{l-1}])]\Big) \subseteq$$
$$Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_{l-1}])),t(\alpha[j_{l-1}])]\Big)$$

und dies für $1<l\leq m$. Damit ist (c) bewiesen.

zu(d):

$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [\min_{i\in\{1,\ldots,l\}} t(WP(\alpha[j_i])),t(x)]\Big)| <$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_k])),t(x)]\Big)|$$

kann nur dann gelten (vgl. Definition 4.2.2/1), wenn:

$$[\min_{i\in\{1,\ldots,l\}} t(WP(\alpha[j_i])),t(x)] \subset [t(WP(\alpha[j_k])),t(x)]$$

erfüllt ist. Dies kann aber nur dann der Fall sein, wenn:

$$\min_{i\in\{1,\ldots,l\}} t(WP(\alpha[j_i]))>t(WP(\alpha[j_k]))$$

und daher für alle $i\in\{1,\ldots,l\}$: $WP(\alpha[j_i])\neq WP(\alpha[j_k])$ gilt.

zu(e): Angenommen $k ::= \min\{i \mid t(WP(\alpha[j_i]))<t(WP(\alpha[j_i])) \;\&\; l<i\leq m\}$. Dann muß, aufgrund der Wahl von k gelten:

$$t(WP(\alpha[j_{k-1}]))\geq t(WP(\alpha[j_l]))$$

Das bedeutet:

$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_k])),t(x)]\Big) -$$
$$Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_k])),t(WP(\alpha[j_{k-1}]))]\Big)| \leq$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_i])),t(x)]\Big)|,$$

und damit folgt:

$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_k])),t(x)]\Big) -$$

$$Z\Big(\bigcup_{\alpha[i]\in W(ax)\&f(\alpha[i])} F_{nv(ax)}(\alpha[i]) \mid [t(WP(\alpha[j_i])), t(x)]\Big)| \leq$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(ax)\&f(\alpha[i])} F_{nv(ax)}(\alpha[i]) \mid [t(WP(\alpha[j_k])), t(WP(\alpha[j_{k-1}]))]\Big)|.$$

Bleibt zu zeigen, daß

$$|Z\Big(\bigcup_{\alpha[i]\in W(ax)\&f(\alpha[i])} F_{nv(ax)}(\alpha[i]) \mid [t(WP(\alpha[j_k])), t(WP(\alpha[j_{k-1}]))]\Big)| < n_1 * n_2$$

gilt. Nach Annahme stehen $\alpha[j_k]$ und $\alpha[j_{k-1}]$ in der Beziehung $\alpha[j_k](E \cup B)\alpha[j_{k-1}]$ zueinander. $\alpha[j_k]E\alpha[j_{k-1}]$ kann nicht gelten, denn in diesem Falle wäre $t(WP(\alpha[j_k]))=t(WP(\alpha[j_{k-1}]))$ und j_{k-1} hätte bereits die Eigenschaft:

$$t(WP(\alpha[j_{k-1}]))<t(WP(\alpha[j_i]))$$

im Widerspruch zur Wahl von k, also muß $\alpha[j_k]B\alpha[j_{k-1}]$ gelten. Das heißt aber:

$$t(\alpha[j_k])>t(\alpha[j_{k-1}])>t(WP(\alpha[j_{k-1}]))\geq t(WP(\alpha[j_i]))>t(WP(\alpha[j_k]))$$

und es kann weiter auf:

$$|Z\Big(\bigcup_{\alpha[i]\in W(ax)\&f(\alpha[i])} F_{nv(ax)}(\alpha[i]) \mid [t(WP(\alpha[j_k])), t(WP(\alpha[j_{k-1}]))]\Big)| \leq$$
$$|Z\Big(\bigcup_{\alpha[i]\in W(ax)\&f(\alpha[i])} F_{nv(ax)}(\alpha[i]) \mid [t(WP(\alpha[j_k])), t(\alpha[j_k])]\Big)| < n_1 * n_2$$

geschlossen werden (zum letzteren "<" vgl. Hilfssatz 4.2.2/1, Aussage (e)), womit auch Punkt (e) bewiesen ist. ■

4.2.3. Begrenzung der vermeidbaren Rücksetzausbreitung

Die Vorüberlegungen in Abschnitt 4.2.2 haben einen Zusammenhang zwischen dem Entstehen des Dominoeffektes und drei wohl zu unterscheidenden Bedingungen hergestellt. Mittels zweier im folgenden zu beweisenden Sätze wird nachgewiesen, daß:

(1) zum einen (Satz 4.2.3/1) bei Nichtvorhandensein der drei in Abschnitt 4.2.2 identifizierten Probleme, bezogen auf alle aktiven Berechnungen α eines Prozeßsystems P, sich die Größe des im Sinne von Definition 4.2.1/1 vermeidbaren Rücksetzaufwandes im Fehlerfall prinzipiell begrenzen läßt, d.h. das Nichtauftreten dieser Situationen ist hinreichend für das Nichtauftreten des Problems eines unbegrenzbar großen vermeidbaren Rücksetzaufwandes, und

(2) zum anderen (Satz 4.2.3/2) Bedingungen abgeleitet werden können, die für das Nichtauftreten des Problems eines unbegrenzbar großen vermeidbaren

Rücksetzaufwandes sowohl hinreichend als auch notwendig sind

Die Überlegungen beziehen sich auf den Fall eines auf eine *einzelne Recoveryeinheit eingrenzbaren Fehlers* Unter Rückgriff auf Hilfssatz 3.4.2/2 lassen sich die abgeleiteten Ergebnisse genau dann auch auf den Fall von *mehreren fehlerhaften Recoveryeinheiten* verallgemeinern, falls, wie in Korollar 4.1/1 festgehalten, die Zahl der maximal in einer beliebigen Fehlersituation auftretenden Fehler, für ein gegebenes Prozeßsystem $P=(A,D,w)$, berechnungsunabhängig abgeschätzt werden kann und des weiteren, wie man sich leicht überlegt, die Zahl der Aktionen, die im Zeitraum zwischen der Entstehung eines Fehlers bis zu seiner Erkennung ausgeführt worden sind und aufgrund des Fehlers zurückgesetzt werden müßten, ebenfalls berechnungsunabhängig beschränkt werden kann.

Bemerkung: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge von P Der Zusatz "Nichtberücksichtigung potentieller Fehlerverursacher" wird im folgenden gleichgesetzt mit der Zulässigkeit der Folgerung:

$$\forall \alpha[i],\alpha[j]\in W(\alpha)\colon [\alpha[j]B_a\alpha[i] \Rightarrow v(\alpha[j]B_a\alpha[i])].$$

Gemäß Definition 3.2.3/3 gilt in diesem Falle:

$$F_{nv(a)}(\alpha[i]) = \{\alpha[i]\},$$

was bei der Interpretation der im folgenden und in den nächsten Abschnitten abgeleiteten Ergebnisse gegebenenfalls entsprechend zu berücksichtigen ist.

Satz 4.2.3/1: Sei αx, $x\in A$, eine beliebige aktive Berechnungsfolge des aus n_4 Prozessen bestehenden Prozeßsystems $P=(A,D,w)$ Wird im Zustand $val_P(\alpha x)$ die Zurücksetzung des Prozesses $P_{bs(x)}$ zum Wiederaufsetzpunkt $WP_{ax}(x)$ erforderlich, so gilt bei Nichtberücksichtigung potentieller Fehlerverursacher (im folgenden wird dort, wo dies nicht zu Verwechslungen führen kann, der Index "αx" bzw. der Index "βy" weggelassen):

$$((1)\ \&\ (2)\ \&\ (3)) \Rightarrow (\exists n\in\mathbf{N}\forall\alpha\colon [\alpha \text{ aktiv in } P \Rightarrow |V_Z(\bigcup_{\alpha[i]\in W(ax)\,\&\,f(\alpha[i])} F_{nv(ax)}(\alpha[i]))|\leq n])$$

wobei:

$$(1)\quad \exists n_1\forall i\in\{1,\ldots,n_4\}\forall\beta\forall y\in A\forall\beta[j]\in Z(\bigcup_{\beta[l]\in W(\beta y)\,\&\,f(\beta[l])} F_{nv(\beta y)}(\beta[l]))\cap P_i\colon$$
$$[(\beta y \text{ aktiv in } P\ \&$$
$$t(\beta[j])=min\{t(\beta[k]) \mid \beta[k]\in Z(\bigcup_{\beta[l]\in W(\beta y)\,\&\,f(\beta[l])} F_{nv(\beta y)}(\beta[l]))\cap P_i\}) \Rightarrow$$

$$\{\exists\beta_{j_1}\ldots\beta_{j_m}\,.\,\beta[j]=\beta_{j_m}[\langle E\cup B\rangle\beta[j_{m-1}]\,.\,\beta[j_2]\langle E\cup B\rangle\beta[j_1]=y \;\&\; \{WP(\beta[j_i])\mid\beta[j_i]\in\{\beta[j_1],\ldots,\beta[j_m]\}\cap P_i \;\&\; \neg\exists\beta[j_k]\in\{\beta[j_1]\,.\,\ldots\,.\,\beta[j_{i-1}]\}\cap P_i\,.\; t(WP(\beta[j_i]))\geq t(WP(\beta[j_k])) \;\&\; \neg up(\beta[j_i])\}|<n_1)];$$

(2) $\exists n_2\forall\beta\forall\beta[j]\in W(\beta)\colon[\beta$ aktiv in $P \Rightarrow |\{i\mid\beta[i]\in W(\beta)\;\&\;\beta[j]E_\beta\beta[i]\}|<n_2]$;

(3) $\exists n_3\forall b\in A\forall\beta_1,\beta_2\colon[(b\in bef_P(\beta_1)\;\&\;\beta_1\beta_2$ aktiv in $P \;\&\; P_{bz(b)}\cap W(\beta_2)=\phi) \Rightarrow |\beta_2|<n_3]$;

und $n_1,n_2,n_3\in\mathbf{N}$

Informell besagt dieser Satz: Bei Korrektur des fehlerhaften Systemzustandes $val_P(\alpha x)$ mittels Rücksetzens kann, unter der Voraussetzung $val_P(\alpha)$ sei ein verifizierter/validierter Zustand, der Dominoeffekt dann nicht auftreten, wenn:

(1) der Umfang der *zyklischen Zurücksetzung* im Fehlerfall, bezogen auf jeden Prozeß, durch eine Konstante n_1 beschränkbar ist,

(2) jede *Recoveryeinheit weniger als* n_2 Aktionen umfaßt, und

(3) *jeder Prozeß*, der nicht beendet ist, *mindestens alle* n_3-1 *Aktionen* rechnend wird, d.h. zumindest eine Aktion ausführt.

Beweis: Nach den Voraussetzungen des Satzes gilt:

$$\bigcup_{\alpha[i]\in W(\alpha x)\;\&\;f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\subseteq\{\alpha[j]\mid\alpha[j]\in W(\alpha x)\;\&\;xE\alpha[j]\}.$$

Unter Berücksichtigung dieser Enthaltenseinsbeziehung und der Aussage von Hilfssatz 4.2.1/1 folgt

$$V_Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big)\subseteq V_Z(x)\subseteq Z(x)=Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big).$$

Damit kann die Beziehung:

$$|V_Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big)|\leq n\leq\prod_{i=1}^{4} n_i$$

nachgewiesen werden, in dem:

$$|Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big)|=|Z(x)|\leq n\leq\prod_{i=1}^{4} n_i$$

für eine berechnungsunabhängige Konstante $n \in \mathbb{N}$ gezeigt wird, was im folgenden geschehen soll:

Angenommen (1), (2) und (3) seien erfüllt, aber es existiere unter den Voraussetzungen des Satzes kein n, so daß gilt:

$$|Z(\bigcup_{\alpha[i] \in W(\alpha x) \& f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))| = |Z(x)| \leq n \leq \prod_{i=1}^{4} n_i$$

für jedes beliebige α und jede beliebige Aktion x, die den Voraussetzungen des Satzes genügen. Dann muß es eine Berechnung α und eine Aktion $x \in A$ geben, αx aktiv, $P_{bz(x)}$ ist zurückzusetzen zu $WP(x)$, so daß gilt:

$$|Z(\bigcup_{\alpha[i] \in W(\alpha x) \& f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))| = |Z(x)| > \prod_{i=1}^{4} n_i .$$

Das heißt aber, es muß eine Aktion $\alpha[j] \in Z(x)$ geben,

$$\alpha[j](E \cup B)^* x \quad \& \quad t(\alpha[j]) \leq t(x),$$

so daß gilt:

$$|Z(\bigcup_{\alpha[i] \in W(\alpha x) \& f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j]), t(x)])| =$$
$$|Z(\bigcup_{\alpha[i] \in W(\alpha x) \& f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))| = |Z(x)| > \prod_{i=1}^{4} n_i .$$

Aufgrund des Hilfssatzes 3.2.4/5 kann von $\alpha[j](E \cup B)^* x$ auf die Existenz von Aktionen $\alpha[j_1], \ldots, \alpha[j_m]$ geschlossen werden, die bezüglich der Relation $(E \cup B)$ wie folgt in Beziehung stehen:

$$(*)\ \alpha[j] = \alpha[j_m](E \cup B)\alpha[j_{m-1}] \ldots \alpha[j_2](E \cup B)\alpha[j_1] = x .$$

Da nach Hilfssatz 4.2.2/1-(d) (unter den Voraussetzungen (*), (2) und (3)) für $\alpha[j_l](E \cup B)\alpha[j_{l-1}]$, $l \in \{2, \ldots, m\}$,

$$|Z(\bigcup_{\alpha[i] \in W(\alpha x) \& f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j_l]), t(\alpha[j_{l-1}])])| \leq$$
$$|Z(\bigcup_{\alpha[i] \in W(\alpha x) \& f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_{l-1}])), t(\alpha[j_{l-1}])])| \leq n_2^* n_3$$

gilt, muß es aufgrund der Beziehung

$$|Z(\bigcup_{\alpha[i] \in W(\alpha x) \& f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j]), t(x)])| = |Z(x)| > \prod_{i=1}^{4} n_i$$

unter den n_4 Prozessen mindestens einen Prozeß P_r geben, dem nicht weniger als n_1 paarweise verschiedene Aktionen $\alpha[j_{l_i}]$, $l_i \in \{1, \ldots, m\}$, $i \in \{1, \ldots, n_1\}$, angehören, wobei für diese gilt (folgt zusammen mit Hilfssatz 4.2.2/2-(d)/(e)):

$(**)\ \forall i\in\{2,\ldots,n_1\}\colon [l_{i-1}<l_i\ \&\ t(WP(\alpha[j_{l_{i-1}}]))>t(WP(\alpha[j_{l_i}]))\ \&\ WP(\alpha[j_{l_{i-1}}])\neq WP(\alpha[j_{l_i}])]$.

Da $(**)$ jedoch einen Widerspruch zur Voraussetzung (1) darstellt, kann die Annahme:

$$\neg\exists n\in\mathbf{N}\colon [\forall\alpha\forall x\in A\colon [(\alpha x \text{ aktiv in } P \ \&\ P_{bz(x)} \text{ ist zurückzusetzen zu } WP(x)) \Rightarrow (|Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nw(\alpha x)}(\alpha[i]))| = |Z(x)| \leq n \leq \prod_{i=1}^{4} n_i)]].$$

unter den Bedingungen (1) bis (3) nicht gültig sein. ■

Will man die *Umkehrung des Satzes 4.2.3/1* beweisen, so muß man drei Fälle unterscheiden und mit (1), (2), (3) wie in Satz 4.2.3/1 die folgende Beziehung nachweisen

$$(\neg(1) \,||\, \neg(2) \,||\, \neg(3)) \Rightarrow (*),$$

wobei

$$(*)\quad \neg\exists n\in\mathbf{N}\colon [\forall\alpha\forall x\in A\colon [(\alpha x \text{ aktiv in } P \ \&\ P_{bz(x)} \text{ ist zurückzusetzen zu } WP(x)) \Rightarrow (|V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nw(\alpha x)}(\alpha[i]))| \leq n \leq \prod_{i=0}^{4} n_i)]].$$

Beim Versuch, den Beweis für diese drei Fälle ($\neg(1)$, $\neg(2)$ und $\neg(3)$) zu erbringen, stellt man sehr schnell fest, daß die Fälle 1 und 2 $((\neg(1) \,||\, \neg(2)) \Rightarrow (*))$ zwar relativ einfach zu beweisen sind, (vgl. Satz 4.2.2/1), Fall 3 $(\neg(3) \Rightarrow (*))$ jedoch im allgemeinen nicht gelten muß.
Als Ausweg aus diesem Dilemma bietet sich die folgende Lösung an, die darin besteht, die Bedingung (3) des Satzes 4.2.3/1 so in eine neue Bedingung abzuändern, daß Fall 3 beweisbar wird und die Aussage von Satz 4.2.3/1 erhalten bleibt.
Ein erster Lösungsversuch führt zu der folgenden Bedingung:

$$(3)'\ \exists n_3\forall b\in A\forall\beta_1,\beta_2\colon [(b\in bef_P(\beta_1)\ \&\ \beta_1\beta_2 \text{ aktiv in } P\ \&\ P_{bz(b)}\cap W(\beta_2)=\phi) \Rightarrow |\{\beta_2[i] \mid \beta_2[i](E_\beta\cup B_\beta)^* b\}| < n_3].$$

Der Nachteil dieser Änderung ist, daß man mit (3)' statt (3) zwar nun in der Lage ist, Fall 3 nachzuweisen, Satz 4.2.3/1 würde jedoch durch diese Änderung seine Gültigkeit verlieren. Dies führt im zweiten Lösungsversuch zu dem Ansatz:

$$(3)''\ \exists n_3\forall\beta[j]\in W(\beta)\forall\beta\colon [(\beta \text{ aktiv in } P) \Rightarrow (|V_{Z_\beta}(\bigcup_{\beta[i]\in W(\beta)\&f_\beta(\beta[i])} F_{nw(\beta)}(\beta[i]) \,|[t(WP_\beta(\beta[j])), t(\beta[j])])| < n_3)].$$

Der Beweis des nachfolgenden Satzes 4.2.3/2 zeigt, daß unter der Bedingung (3)'' (statt (3)) sowohl Fall 3 als auch die Entsprechung zu Satz 4.2.3/1 bewiesen werden können. Des weiteren macht (3)'' (im nachfolgenden Satz 4.2.3/2 mit (2) bezeichnet) die Forderung (2) des Satzes 4.2.3/1 überflüssig.

Satz 4.2.3/2: Sei αx eine beliebige aktive Berechnungsfolge des aus n_3 Prozessen bestehenden Prozeßsystems $P=(A,D,\mathcal{W})$. Wird (z.B. nach Ausführung eines Akzeptanztests $AT_{bz(x),k}$ im Zustand $val_P(\alpha)$ durch die akzeptanztestausführende Aktion x aufgrund eines durch $AT_{bz(x),k}$ erkannten Fehlers) im Zustand $val_P(\alpha x)$ die Zurücksetzung des Prozesses $P_{bz(x)}$ zum Wiederaufsetzpunkt $WP_{\alpha x}(x)$ erforderlich, so gilt bei Nichtberücksichtigung potentieller Fehlerverursacher (im folgenden wird dort, wo dies nicht zu Verwechslungen führen kann, der Index "αx", der Index "βy" bzw. der Index "β" weggelassen):

$$((1)\ \&\ (2))\ \Leftrightarrow\ (\exists n\in\mathbf{N}\forall\alpha\colon[\alpha \text{ aktiv in } P\ \Rightarrow\ |V_Z(\bigcup_{\alpha[i]\in\mathcal{W}(\alpha x)\&f(\alpha[i])}F_{nv(\alpha x)}(\alpha[i]))|\leq n]),$$

wobei:

(1) $\exists n_1\forall i\in\{1,\ldots,n_4\}\forall\beta\forall y\in A\forall\beta[j]\in Z(\bigcup_{\beta[l]\in\mathcal{W}(\beta y)\&f(\beta[l])}F_{nv(\beta y)}(\beta[l]))\cap P_i$:

$[(\beta y$ aktiv in P &

$t(\beta[j])=min\{t(\beta[k])\mid\beta[k]\subset Z(\bigcup_{\beta[l]\in\mathcal{W}(\beta y)\&f(\beta[l])}F_{nv(\beta y)}(\beta[l]))\cap P_i\}\ \Rightarrow$

$(\exists\beta[j_1],\ldots,\beta[j_m]\colon[\beta[j]=\beta[j_m](E\cup B)\beta[j_{m-1}]\ldots\beta[j_2](E\cup B)\beta[j_1]=y$

$\&\ |\{\mathcal{WP}(\beta[j_i])\mid\beta[j_i]\in\{\beta[j_1],\ldots,\beta[j_m]\}\cap P_i\ \&\ \neg\exists\beta[j_k]\in\{\beta[j_1],\ldots,\beta[j_{i-1}]\}\cap P_i$:

$\&\ t(\mathcal{WP}(\beta[j_i]))\geq t(\mathcal{WP}(\beta[j_k]))$

$\&\ \neg wp(\beta[j_i])\}|<n_1])]$;

(2) $\exists n_2\forall\beta\forall\beta[j]\in\mathcal{W}(\beta)\colon[(\beta$ aktiv in $P)\ \Rightarrow$

$(|V_Z(\bigcup_{\beta[i]\in\mathcal{W}(\beta)\&f(\beta[i])}F_{nv(\beta)}(\beta[i])\mid[t(\mathcal{WP}(\beta[j])),t(\beta[j])])|<n_2)]$.

Beweis: Nach den Voraussetzungen des Satzes gilt:

$$\bigcup_{\alpha[i]\in\mathcal{W}(\alpha x)\ \&\ f(\alpha[i])}F_{nv(\alpha x)}(\alpha[i])\subseteq\{\alpha[j]\mid\alpha[j]\in\mathcal{W}(\alpha x)\ \&\ xE_{\alpha x}\alpha[j]\}.$$

Damit folgt aber

$$V_Z(\bigcup_{\alpha[i]\in\mathcal{W}(\alpha x)\&f(\alpha[i])}F_{nv(\alpha x)}(\alpha[i]))\subseteq V_Z(x)$$

und es genügt zu zeigen, daß ein n existiert, so daß $|V_Z(x)|\leq n\leq\prod_{i=1}^{3}n_i$, unabhängig von x

und der Berechnung αx

"=>": Angenommen (1) und (2) seien erfüllt, aber unter den Voraussetzungen des Satzes existiere kein n, so daß gilt:

$$|V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))| \leq |V_Z(x)| \leq n \leq \prod_{i=1}^{3} n_i$$

für ein beliebiges α und eine beliebige Aktion $x\in A$. Dann muß es eine Berechnung α und eine (akzeptanztestausführende) Aktion x geben, so daß unter den Annahmen des Satzes gilt:

$$|V_Z(x)| \geq |V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))| > \prod_{i=1}^{3} n_i.$$

Das heißt aber, es muß eine Aktion $\alpha[j]\in V_Z(x)$ geben,

$$\alpha[j](E\cup B)^* x \ \&\ t(\alpha[j])\leq t(x),$$

so daß gilt:

$$|V_Z((x) \quad [t(\alpha[j]),t(x)])| \geq |V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j]),t(x)])| > \prod_{i=1}^{3} n_i.$$

Aufgrund des Hilfssatzes 3.2.4/5 kann von $\alpha[j](E\cup B)^* x$ auf die Existenz von Aktionen $\alpha[j_1], \ldots, \alpha[j_m]$ geschlossen werden, die bezüglich der Relation $(E\cup B)$ wie folgt in Beziehung stehen:

$$(*)\ \alpha[j]=\alpha[j_m](E\cup B)\alpha[j_{m-1}]\ldots\alpha[j_2](E\cup B)\alpha[j_1]=x.$$

Da nach (2) für $\alpha[j_l](E\cup B)\alpha[j_{l-1}]$, $l\in\{2,\ldots,m\}$,

$$|V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j_l])),t(\alpha[j_{l-1}])])| < n_2$$

(denn mit $\alpha[j_l](E\cup B)\alpha[j_{l-1}]$ folgt *entweder*:

$$\alpha[j_l]E\alpha[j_{l-1}] \text{ und damit: } t(WP(\alpha[j_l]))=t(WP(\alpha[j_{l-1}]))$$

oder:

$$\alpha[j_l]B\alpha[j_{l-1}], \text{ d.h. } t(\alpha[j_l])>t(\alpha[j_{l-1}]),$$

und somit:

$$[t(WP(\alpha[j_l])),t(\alpha[j_{l-1}])] \subset [t(WP(\alpha[j_l])),t(\alpha[j_l])];)$$

gilt, muß es aufgrund der Beziehung

$$|V_Z((x) \mid [t(\alpha[j]),t(x)])| \geq |V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(\alpha[j]),t(x)])| > \prod_{i=1}^{3} n_i$$

unter den n_3 Prozessen von P mindestens einen Prozeß P_r geben, dem nicht weniger als n_1 paarweise verschiedene Aktionen $\alpha[j_{i_l}]$, $l_i\in\{1,\ldots,m\}$, $i\in\{1,\ldots,n_1\}$, angehören,

wobei für diese gilt:

$$(**)\ \forall i\in\{2,\ldots,n_1\}\colon [l_{i-1}<l_i\ \&\ t(WP(\alpha[j_{l_{i-1}}]))>t(WP(\alpha[j_{l_i}]))\ \&\ WP(\alpha[j_{l_{i-1}}])\neq WP(\alpha[j_{l_i}])]$$

Da (**) jedoch im Widerspruch zur Voraussetzung (1) steht, muß die Annahme:

$$\neg\exists n\in\mathbf{N}\colon [\forall\alpha\forall x\in A\colon [(\alpha x \text{ aktiv in } P\ \&\ P_{bz(x)} \text{ ist zurückzusetzen zu } WP(x)) \Rightarrow (|V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))|\leq n\leq \prod_{i=1}^{3} n_i)]]$$

falsch sein.

"<=": Es muß gezeigt werden, daß unter den Voraussetzungen des Satzes gilt:

$$(\neg(1)\ \|\ \neg(2)) \Rightarrow (\neg\exists n\in\mathbf{N}\forall\alpha\forall x\in A\colon [(\alpha x \text{ aktiv in } P\ \&\ P_{bz(x)} \text{ ist zurückzusetzen zu } WP(x)) \Rightarrow (|V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))|\leq n\leq \prod_{i=1}^{3} n_i)].$$

Im folgenden wird dies nachgewiesen, indem die Annahme *"ein passendes n existiere"* in beiden zu unterscheidenden Fällen widerlegt wird:

Fall 1 ($\neg(1) \Rightarrow \ldots$). In diesem Falle muß es eine Berechnungsfolge αx geben sowie Aktionen $\alpha[j_1],\ldots,\alpha[j_m]$ und einen Prozeß P_r, so daß gilt:

$$\alpha[j_m](E\cup B)\alpha[j_{m-1}]\ldots\alpha[j_2](E\cup B)\alpha[j_1]=x\ \&\ |\{WP(\alpha[j_i])\mid \alpha[j_i]\in\{\alpha[j_1],\ldots,\alpha[j_m]\}\cap P_r\ \&\ \neg\exists\alpha[j_k]\in\{\alpha[j_1],\ldots,\alpha[j_{i-1}]\}\colon [t(WP(\alpha[j_i]))\geq t(WP(\alpha[j_k]))\ \&\ \neg up(\alpha[j_i])]\}|>n+2,$$

d.h. es muß eine Kette von mindestens $n+2$ verschiedenen Aktionen $\alpha[j_{i_k}]\in\{\alpha[j_2],\ldots,\alpha[j_m]\}$, $i\in\{1,\ldots,n+2\}$, geben, so daß gilt:

$$(*)\ 0<l_1<\ \ldots\ <l_{n+2}\ \&\ t(WP(\alpha[j_{i_{n+2}}]))<t(WP(\alpha[j_{i_{n+1}}]))<\ldots<t(WP(\alpha[j_{i_1}]))<t(WP(x))$$

und

$$(**)\ \forall i\in\{2,\ldots,n+2\}\colon\ \neg up(\alpha[j_{i_k}])\ \&\ t(\alpha[j_{i_k}])<t(WP(x))].$$

D.h. aber, daß aufgrund von (*) und (**) im angenommenen Fehlerfall mindestens $n+1$ Aktionen des Prozesses P_r zurückgesetzt werden müßten, deren Zurücksetzung vermeidbar wäre, da sie allesamt bereits vor dem Zeitpunkt $t(WP(x))$ beendet wurden.

Fall 2 ($\neg(2) \Rightarrow \ldots$). In diesem Falle muß es in P eine aktive Berechnungsfolge αx geben, $val_P(\alpha)$ verifiziert/validiert, und $val_P(\alpha x)$ fehlerhaft, so daß für ein $\alpha[j]\in W(\alpha)$ gilt:

$$(*)\quad |V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\mid [t(WP(\alpha[j])),t(\alpha[j])])|>n.$$

Aufgrund der Beziehung:

$$V_Z(\bigcup_{\alpha[i]\in W(\alpha x)\&f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))\supseteq$$

$$V_Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\,\&\,f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]) \mid [t(WP(\alpha[j])), t(\alpha[j])]\Big),$$

folgt mit (*):

$$|V_Z\Big(\bigcup_{\alpha[i]\in W(\alpha x)\,\&\,f(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big)| > n.$$

Damit ist die Annahme *"ein passendes n existiere"* auch im Fall 2 widerlegt. ■

Das Problem der *unbeschränkbar langen Prozeßinaktivphasen*, als eine der Ursachen für den Dominoeffekt, läßt sich reduzieren auf das Problem der *unbeschränkbar langen Prozeßinaktivphasen innerhalb einer Recoveryeinheit*. Zwischen zwei Recoveryeinheiten können unbeschränkbar lange Prozeßinaktivphasen bestehen.
Wird Satz 4.2.3/1 entsprechend dieser Erkenntnis in Punkt (3) modifiziert zu:

(3) $\exists n_3\in\mathbf{N}\,\forall\beta\,\forall\beta'[i],\beta[j]\in W(\beta).\ [\beta[i]E_{\beta}\beta[j]\ \&\ \neg\exists k\colon [i<k<j\ \&\ \beta[i]E_{\beta}\beta[k]] \Rightarrow |(\beta|[t(\beta[i]), t(\beta[j])])| < n_3]$

bleibt die Gültigkeit der Aussage von Satz 4.2.3/1 erhalten, wenn man den Beweis von Punkt (b) des Hilfssatzes 4.2.2/1 wie folgt ändert (mit n_2 statt n_3):

> Da $(t\alpha[j_i])\leq t(\alpha[j_{i-1}])$ und nach Annahme $\alpha[j_i](E\cup B)\alpha[j_{i-1}]$ gilt, muß $\alpha[j_i]E\alpha[j_{i-1}]$ erfüllt sein.
> Sei o.B.d.A. $bz(\alpha[j_i])=i$. Unter der Voraussetzung (1) können im Prozeß P_i im Zeitintervall $[t(\alpha[j_i]), t(\alpha[j_{i-1}])]$ maximal n_1-1 Aktionen und, unter Voraussetzung (3) (siehe oben), in den restlichen Prozessen von P maximal $(n_1-2)*(n_2-1)$ Aktionen ausgeführt worden sein. Insgesamt können im Intervall $[t(\alpha[j_i]), t(\alpha[j_{i-1}])]$ also nicht mehr als:
>
> $$(n_1-1)+(n_1-2)*(n_2-1) < n_1*n_2$$
>
> Aktionen ausgeführt worden sein.

Die modifizierte Fassung von Satz 4.2.3/1 läßt sich wie folgt interpretieren:

> Bestehen alle Recoveryeinheiten eines Prozeßsystems aus beschränkbar vielen atomaren Aktionen und ist die Ausführungsdauer aller Recoveryeinheiten beschränkbar, so ist, bei Vermeidung des Problems der unbeschränkbaren zyklischen Zurücksetzung, das Prozeßsystem in der in Satz 4.2.3/1 angenommenen Fehlersituation dominoeffektfrei.

4.2.4. Begrenzung der Zahl zurückzusetzender Aktionen

In den Sätzen 4.2.3/1 und 2 des vorangehenden Abschnittes wurde gezeigt, unter welchen Bedingungen sich die Zahl der vermeidbar zurückzusetzenden Aktionen einer Berechnung im Falle von auf eine Recoveryeinheit begrenzten Fehlern beschränken läßt.
Die im folgenden notierten beiden Sätze verallgemeinern diese Ergebnisse auf die Zahl der insgesamt, d.h. sowohl vermeidbar als auch unvermeidbar, zurückzusetzenden Aktionen.
Satz 4.2.4/1 zeigt hinreichende Bedingungen auf, unter welchen sich die Zahl der zurückzusetzenden Aktionen in einer auf eine einzige Recoveryeinheit begrenzten Fehlersituation beschränken läßt. Satz 4.2.4/2 identifiziert für diesen Fehlerfall Bedingungen, von denen gezeigt werden kann, daß sie nicht nur hinreichend, sondern auch notwendig sind.
Die Ableitung von Hilfssatz 4.2.4/1 erlaubt schließlich eine Verallgemeinerung der Aussagen der Sätze 4.2.4/1 und 2 hinsichtlich der angenommenen Fehlersituation in Korollar 4.2.4/1 bzw. Korollar 4.2.4/2.

Satz 4.2.4/1: Unter den allgemeinen Voraussetzungen von Satz 4.2.3/1 gilt:

$$((1)\ \&\ (2)\ \&\ (3)) \Rightarrow (\exists n \in \mathbb{N} \forall \alpha: [\alpha \text{ aktiv in } P \Rightarrow |Z_{ax}(\bigcup_{\alpha[i] \in W(ax) \& f_{ax}(\alpha[i])} F_{nv(ax)}(\alpha[i]))| \leq n]),$$

wobei (1), (2) und (3) wie in Satz 4.2.3/1 definiert sind.

Beweis: Wie man sich leicht überzeugt, stellt der Beweis zu Satz 4.2.3/1 auch einen Beweis für die Aussage von Satz 4.2.4/1 dar. ■

Satz 4.2.4/2: Unter den allgemeinen Voraussetzungen des Satzes 4.2.3/2 gilt:

$$((1)\ \&\ (2)) \Leftrightarrow (\exists n \in \mathbb{N} \forall \alpha: [\alpha \text{ aktiv in } P \Rightarrow |Z_{ax}(\bigcup_{\alpha[i] \in W(ax) \& f_{ax}(\alpha[i])} F_{nv(ax)}(\alpha[i]))| \leq n]),$$

wobei (2) wie folgt und (1) wie in Satz 4.2.3/2 definiert ist:

$$(2)\ \exists n_2 \in \mathbb{N} \forall \beta \forall \beta[j] \in W(\beta): [(\beta \text{ aktiv in } P) \Rightarrow (|Z_\beta(\bigcup_{\beta[i] \in W(\beta) \& f_\beta(\beta[i])} F_{nv(\beta)}(\beta[i]) \mid [t(WP_\beta(\beta[j])), t(\beta[j])])| < n_2)].$$

Beweis:

"=>": Analog zu Satz 4.2.3/2.

"<=": Fall 2 ist analog zu Satz 4.2.3/2, Fall 2, beweisbar, und der Fall 1 vereinfacht sich wie folgt:

Fall 1 ($\neg(1) \Rightarrow \ldots$): In diesem Falle muß es eine Berechnungsfolge αx geben sowie Aktionen $\alpha[j_1],\ldots,\alpha[j_m]$, so daß gilt:

$$\begin{aligned} & \alpha[j_m](E\cup B)\alpha[j_{m-1}]\ldots\alpha[j_2](E\cup B)\alpha[j_1]=x \;\;\&\\ & |\{WP(\alpha[j_i]) \mid \alpha[j_i]\in\{\alpha[j_1],\ldots,\alpha[j_m]\} \;\&\; \neg\exists\alpha[j_k]\in\{\alpha[j_1],\ldots,\alpha[j_{i-1}]\}:\\ & \quad [t(WP(\alpha[j_i]))\geq t(WP(\alpha[j_k])) \;\&\; \neg up(\alpha[j_i])]\}| > n. \end{aligned}$$

Damit gilt jedoch $|\{\alpha[j_1],\ldots,\alpha[j_m]\}|>n$, und, da:

$$\{\alpha[j_1],\ldots,\alpha[j_m]\}\subseteq Z_{\alpha x}\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f_{\alpha x}(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big),$$

folgt:

$$|Z_{\alpha x}\Big(\bigcup_{\alpha[i]\in W(\alpha x)\&f_{\alpha x}(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])\Big)| > n,$$

was zu beweisen war. ■

Hilfssatz 4.2.4/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Dann gilt:

$$\begin{aligned} ((1)\;\&\;(2)) \;\Leftrightarrow\; & (\exists n\in\mathbf{N}\forall\alpha\colon [\alpha \text{ aktiv in } P \;\Rightarrow\\ & \forall\alpha[i]\in W(\alpha)\colon [f_\alpha(\alpha[i])=\mathbf{true} \Rightarrow |U_{Z_\alpha}(\alpha[i])|\leq n]]), \end{aligned}$$

wobei (1) und (2) wie folgt definiert sind:

(1) $\exists n_1\in\mathbf{N}\forall\alpha\forall\alpha[i]\in W(\alpha)\colon [\alpha \text{ aktiv in } P \;\&\; f_\alpha(\alpha[i])=\mathbf{true} \Rightarrow |\{\alpha[j] \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[j]B^*_\alpha\alpha[i]\}|\leq n_1]$,

(2) $\exists n_2\in\mathbf{N}\forall\alpha\colon [\alpha \text{ aktiv in } P \Rightarrow |\{\alpha[j] \mid \alpha[j],\alpha[k]\in W(\alpha) \;\&\; \alpha[j]E_\alpha\alpha[k]\}|\leq n_2]$.

Beweis: Folgt unmittelbar mit Definition 4.2.1/1 und den beiden Voraussetzungen (1) und (2). ■

Korollar 4.2.4/1: Sei $P=(A,D,w)$ ein Prozeßsystem. Dann gilt bei Nichtberücksichtigung potentieller Fehlerverursacher und bei Erfülltsein der Bedingungen (1) bis (3) des Satzes 4.2.4/1 sowie der Bedingung (1) des Hilfssatzes 4.2.4/1:

$$\exists n\in\mathbf{N}\forall\alpha\colon [\alpha \text{ aktiv in } P \Rightarrow |Z_\alpha\Big(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i])\Big)|\leq n].$$

Beweis: Ist eine unmittelbare Folgerung der Aussage des Satzes 4.2.4/1 und von Hilfssatz 4.2.4/1. ■

Korollar 4.2.4/2: Sei $P=(A,D,w)$ ein Prozeßsystem. Dann gilt bei Nichtberücksichtigung potentieller Fehlerverursacher, daß die Bedingungen (1) und (2) des Satzes 4.2.4/2 sowie die Bedingung (1) des Hilfssatzes 4.2.4/1 genau dann erfüllt sind, wenn ein $n \in \mathbf{N}$ existiert, so daß für alle aktiven Abläufe α von P die Größe der Menge:

$$Z_\alpha\Big(\bigcup_{\alpha[i]\in V(\alpha)\wedge f_\alpha(\alpha[i])} F_{w(\alpha)}(\alpha[i])\Big)$$

der bei Rücksetzung aller als fehlerhaft erkannten Aktionen zurückzusetzenden Aktionen durch n beschränkbar ist.

Beweis: Folgt mit der Aussage des Satzes 4.2.4/2 und des Hilfssatzes 4.2.1/1. ■

4.3. Dominoeffekt und Fehlerausbreitung

Im Falle der Nichtberücksichtigung potentieller Fehlerverursacher reicht die Begrenzbarkeit der Rücksetzausbreitung im Fehlerfall zum Ausschluß des Auftretens eines Dominoeffektes aus. Sobald jedoch die potentiellen Verursacher eines Fehlers Berücksichtigung finden sollen, ist für die Dominoeffektfreiheit eines Prozeßsystems zusätzlich auch die Beschränkbarkeit der Fehlerausbreitung Voraussetzung, d.h. die Existenz einer Konstanten $n \in \mathbb{N}$, so daß für jede beliebige fehlerhafte Aktion $\alpha[i]$ einer beliebigen Berechnung α eines Prozeßsystems $P=(A,D,w)$ gilt: $|F_{nv(\alpha)}(\alpha[i])| \leq n$.

Die Herleitung von Kriterien, die Aussagen über das Ausmaß einer möglichen Fehlerausbreitung zu formulieren und zu beweisen gestatten, bildet den Schwerpunkt der Untersuchungen dieses Abschnitts. Nach einigen Vorüberlegungen, die es ermöglichen Eigenschaften, die prinzipiell zu einer nicht a priori beschränkbaren Fehlerausbreitung führen können, zu benennen, werden Bedingungen abgeleitet, welche den Umfang der Fehlerausbreitung, gemessen als Zahl der potentiellen Verursacher eines Fehlers, berechnungsunabhängig einzugrenzen erlauben. Eine Verfeinerung des Modellierungsgrades führt schließlich zur Detaillierung der Ergebnisse auf Variablenebene.

4.3.1. Vorüberlegungen zur Fehlerausbreitung

Setzt man sich mit dem Phänomen der Fehlerausbreitung näher auseinander, so erkennt man relativ schnell, daß insbesondere das Auftreten der folgenden drei Probleme zu einer nicht a priori begrenzbaren Fehlerausbreitung führen kann. Im einzelnen handelt es sich um:

(1) das Problem der *unbeschränkbaren Zahl von (direkten) nichtverifizierten Abhängigkeitsbeziehungen einer Aktion*,

(2) das Problem der *unbeschränkbaren Weitergabe nichtverifizierter Variablenwerte* und

(3) das Problem der *unbeschränkbar langen Verifikationslücken*.

Die getroffene Kurzcharakterisierung dieser drei Probleme läßt sich wie folgt umschreiben bzw. bzgl. (2) und (3) anhand der Bilder 4.3.1/1 bzw. 4.3.1/2 veranschaulichen:

zu(1): Das Problem der *unbeschränkbaren Zahl von (direkten) nichtverifizierten Abhängigkeitsbeziehungen einer Aktion* (1) tritt in einem Prozeßsystem P genau dann auf, wenn es in P aktive Berechnungsfolgen α und Aktionen $\alpha[i] \in W(\alpha)$ gibt, so daß die Zahl n der nichtverifizierten Aktionen $\alpha[j] \in W(\alpha)$, auf welchen $\alpha[i]$ basiert, d.h. von denen sie direkt abhängt, nicht unabhängig von α vorab beschränkt werden kann, kurz:

$$\neg \exists n \forall \alpha: [\alpha \text{ aktiv in } P \Rightarrow |\{j \mid \alpha[j] \in W(\alpha) \;\&\; \alpha[i] B_a \alpha[j] \;\&\; \neg \exists \alpha[k] \in W(\alpha): [v(\alpha[k] B_a \alpha[j]) \;\&\; at(\alpha[k])]\}| < n],$$

oder wenn die Zahl der nichtverifizierten Aktionen, von welchen $\alpha[i]$ logisch nicht zu trennen ist, mit denen sie also eine Recoveryeinheit bildet, nicht unabhängig von α beschränkt werden kann, kurz:

$$\neg \exists n \forall \alpha: [\alpha \text{ aktiv in } P \Rightarrow |\{j \mid \alpha[j] \in W(\alpha) \;\&\; \alpha[i] E_a \alpha[j] \;\&\; t(\alpha[i]) > t(\alpha[j]) \;\&\; \neg \exists \alpha[k] \in W(\alpha): [v(\alpha[k] B_a \alpha[j]) \;\&\; at(\alpha[k])]\}| < n].$$

zu(2): Das Problem der *unbeschränkbaren Weitergabe nichtverifizierter Variablenwerte*, oder anders ausgedrückt, das Problem der *Fortpflanzung eines Fehlers über eine nicht vorab beschränkbare Anzahl von Recoveryeinheiten hinweg* (2) ist in einem Prozeßsystem P genau dann gegeben (vgl. Bild 4.3.1/1), wenn es kein $n \in \mathbb{N}$ gibt, so daß für alle aktiven Berechnungsfolgen α von P, Aktionen $\alpha[j], \alpha[j_1], \ldots, \alpha[j_n] \in W(\alpha)$, die in der Beziehung

$$\alpha[j] B_a \alpha[j_1] B_a \alpha[j_2] \ldots \alpha[j_{n-1}] B_a \alpha[j_n]$$

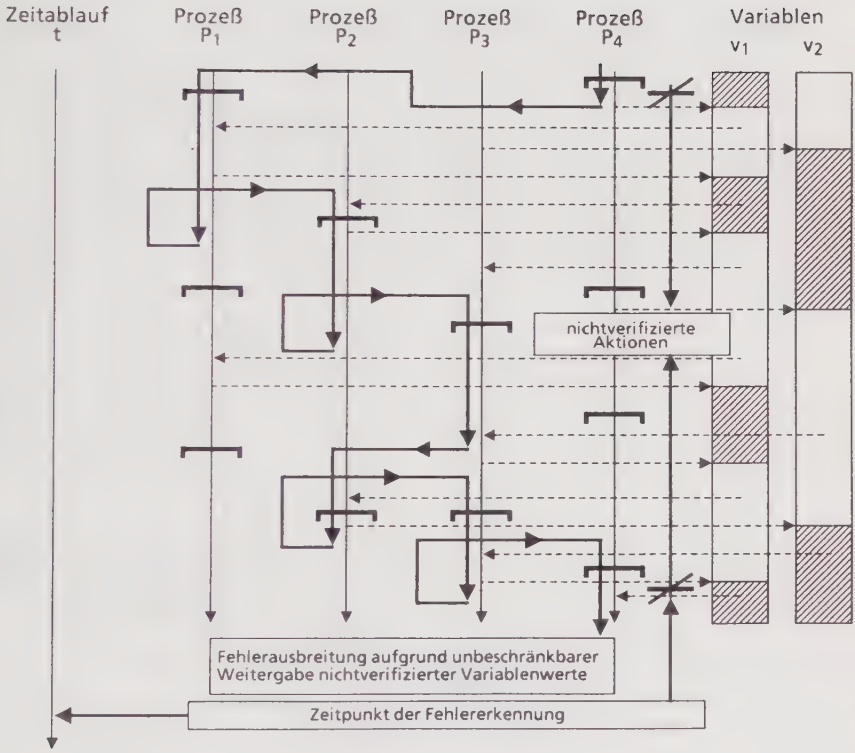
stehen, und alle $i \in \{1, \ldots, n\}$ gilt:

$$(\neg \exists \alpha[k_i] \in W(\alpha): [v(\alpha[k_i] B_a \alpha[j_i]) \;\&\; at(\alpha[k_i])]) \Rightarrow (|\{WP_a(\alpha[j_i]) \mid \alpha[j_i] \in \{\alpha[j_1], \ldots, \alpha[j_n]\}\}| < n).$$

zu(3): Das Problem der *unbeschränkbar langen Verifikationslücken* (3), das als Problem der *unbeschränkbaren Anzahl von Berechnungsschritten von der Änderung eines Variablenwertes bis zu seiner Verifikation* charakterisiert werden kann, besteht in einem Prozeßsystem genau dann (vgl. Bild 4.3.1/2), wenn es kein $n \in \mathbb{N}$ gibt, so daß für zwei Aktionen $\alpha[i]$ und $\alpha[j]$ einer jeden aktiven Berechnungsfolge α von P, die in der Beziehung $\alpha[i] B_a \alpha[j]$ stehen, gilt:

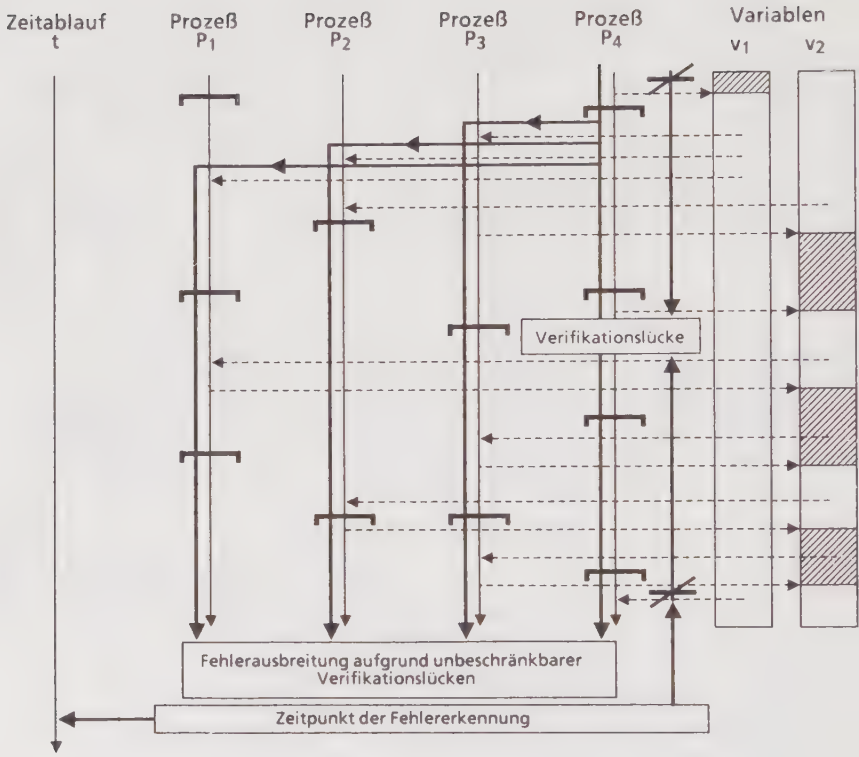
$$(|\{\alpha \mid [t(\alpha[j]), t(\alpha[i])]\}| > n) \Rightarrow (\exists \alpha[k] \in W(\alpha) \;\; [v(\alpha[k] B_a \alpha[j]) \;\&\; at(\alpha[k]) \;\&\; |\{\alpha \mid [t(\alpha[j]), t(\alpha[k])]\}| < n])$$

Anhand der Beschreibung der Probleme (1) bis (3), sowie der zu (2) und (3) skizzierten Bilder 4.3.1/1 bzw. 2, ist leicht zu erkennen, daß diese das Auftreten einer nicht a priori berechnungsunabhängig begrenzbaren Fehlerausbreitung begünstigen und



Legende: siehe Bild 1 3/1 und 3;

Bild 4.3.2/1: Veranschaulichung des Problems der unbeschränkbaren Weitergabe nichtverifizierter Variablenwerte



Legende: siehe Bild 1.3/1 und 3;

Bild 4.3.2/2: Veranschaulichung des Problems der unbeschränkbaren Verifikationslücken

dafur hinreichend sind.
Auf der anderen Seite läßt sich, wie in den beiden folgenden Abschnitten nachgewiesen wird, zeigen, daß bei Nichtauftreten der Probleme (1) bis (3) innerhalb eines Prozeßsystems $P=(A,D,w)$ eine Konstante $n\in\mathbf{N}$ existiert, durch welche jede in P auftretende Fehlerausbreitung beschränkt werden kann, d.h. das Auftreten der Probleme (1) bis (3) ist sowohl notwendig als auch hinreichend für das Auftreten einer unbeschränkbaren Fehlerausbreitung.
Zu beachten ist jedoch, daß die Probleme (1) bis (3) nicht voneinander unabhängig sind, so daß bereits, wie noch zu zeigen sein wird, Untermengen von (1) bis (3) hinreichende und notwendige Bedingungen umfassen.
Des weiteren bleibt anzumerken, daß sich Problem (1) in den in dieser Arbeit betrachteten Typen von Prozeßsystemen $P=(A,D,w)$ nur bezogen auf die Größe einer Recoveryeinheit stellen kann. Bzgl. der Größe der Menge

$$\{j \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[i]B_a\alpha[j]\}$$

gilt für Aktionen $\alpha[i]$ einer jeden beliebigen aktiven Berechnung α:

$$(m\leq|A|) \;\|\; (m\leq|V_P|),$$

wobei:

$$m ::= |\{j \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[i]B_a\alpha[j]\}| \geq$$
$$|\{j \mid \alpha[j]\subset W(\alpha) \;\&\; \alpha[i]B_a\alpha[j] \;\&\; \neg\exists\alpha[k]\in W(\alpha)\colon [v(\alpha[k]B_a\alpha[j]) \;\&\; at(\alpha[k])]\}|,$$

so daß die berechnungsunabhängige Beschränkbarkeit der Menge:

$$\{j \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[i]B_a\alpha[j] \;\&\; \neg\exists\alpha[k]\in W(\alpha)\colon [v(\alpha[k]B_a\alpha[j]) \;\&\; at(\alpha[k])]\} \subseteq$$
$$\{j \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[i]B_a\alpha[j]\}$$

im folgenden gar nicht explizit gefordert werden müßte.

4.3.2. Begrenzung der Fehlerausbreitung

Unter Berücksichtigung der in dem vorangehenden Abschnitt skizzierten Probleme, werden im folgenden Bedingungen abgeleitet, welche den Umfang einer Fehlerausbreitung a priori berechnungsunabhängig abzuschätzen erlauben. Die Ausführungen konzentrieren sich zunächst auf das Zusammenstellen hinreichender Bedingungen in den Sätzen 4.3.2/1 und 2. Im zweiten Schritt werden dann im Rahmen von Satz 4.3.2/3 sowohl hinreichende als auch notwendige Bedingungen benannt.

Mittels eines Induktionsbeweises läßt sich die folgende Aussage, welche die Identifizierung dreier, ausgehend von den in Abschnitt 4.3.1 skizzierten Probleme (1) und (2) abgeleiteten Bedingungen für die Beschränkbarkeit der Fehlerausbreitung zum Inhalt hat, beweisen:

Satz 4.3.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem, das den folgenden drei Bedingungen genügt.

(1) $\exists n_1 \forall \alpha \forall \alpha[j] \in W(\alpha)$: [$\alpha$ aktiv in P => ($|\{i \mid \alpha[i] \in W(\alpha) \;\&\; \alpha[j] E_\alpha \alpha[i]\}| < n_1$)]

(2) $\exists n_2 \forall \alpha \forall \alpha[j] \in W(\alpha)$: [$\alpha$ aktiv in P => ($|\{i \mid \alpha[i] \in W(\alpha) \;\&\; \alpha[j] B_\alpha \alpha[i]\}| < n_2$)].

(3) $\exists n_3 \forall \alpha \forall \alpha[j], \alpha[j_1], \ldots, \alpha[j_n] \in W(\alpha)$: [$\alpha$ aktiv in P =>
[$(\alpha[j] B_\alpha \alpha[j_1] B_\alpha \alpha[j_2] \ldots \alpha[j_{n-1}] B_\alpha \alpha[j_n]$ &
$\forall i \in \{1,\ldots,n\}$: [$\neg \exists \alpha[k_i] \in W(\alpha)$: [$v(\alpha[k_i] B_\alpha \alpha[j_i])$ & $at(\alpha[k_i])$]]
=> ($|\{WP_\alpha(\alpha[j_i]) \mid \alpha[j_i] \in \{\alpha[j_1], \ldots, \alpha[j_n]\}\}| < n_3$)]].

wobei $n_1, n_2, n_3 \in \mathbf{N}$

Unter der Annahme der Gültigkeit dieser drei Bedingungen läßt sich zeigen

$$\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \Rightarrow (|F_{nv(\alpha)}(\alpha[j])| < n)].$$

Beweis: Ist α aktiv in P und wird $\alpha[j] \in W(\alpha)$ als fehlerhaft erkannt, so muß aufgrund der Voraussetzung (3) gelten:

$$F_{nv(\alpha)}(\alpha[j]) \subseteq \bigcup_{i=0}^{n_3-1} F^i_{nv(\alpha)}(\alpha[j]),$$

wobei:

$$F^0_{nv(\alpha)}(\alpha[j]) ::= \{\alpha[j_i] \mid \alpha[j] B_\alpha \alpha[j_i] \;\&\; \alpha[j] E_\alpha \alpha[j_i]\}$$

und

$$F^{i+1}_{nv(\alpha)}(\alpha[j]) ::= \{\alpha[j_i] \mid \exists \alpha[j_q] \in F^i_{nv(\alpha)}(\alpha[j]): [(\alpha[j_q] B_\alpha \alpha[j_i] \;\|\; \exists \alpha[j_k] \in W(\alpha): [\alpha[j_q] B_\alpha \alpha[j_k] E_\alpha \alpha[j_i] \;\&\; \alpha[j_k] B_\alpha \alpha[j_i]]) \;\&\; \alpha[j_i] \notin \bigcup_{r=0}^{i} F^r_{nv(\alpha)}(\alpha[j])]\}.$$

Da somit

$$|F_{nv(\alpha)}(\alpha[j])| \leq |\bigcup_{i=0}^{n_3-1} F^i_{nv(\alpha)}(\alpha[j])| \leq \sum_{i=0}^{n_3-1} |F^i_{nv(\alpha)}(\alpha[j])|,$$

genügt es zu zeigen, daß:

$$\sum_{i=0}^{n_3-1} |F^i_{nv(\alpha)}(\alpha[j])| < n .$$

und dazu muß die Größe der Mengen $F^i_{nv(\alpha)}(\alpha[j])$ abgeschätzt werden. Dies geschieht im folgenden mit Hilfe der vollständigen Induktion:

Induktionsanfang: Unter der Voraussetzung (1) gilt $|F^0_{nv(\alpha)}(\alpha[j])| < n_1$.

Induktionsannahme: $|F^i_{nv(\alpha)}(\alpha[j])| < n_1 * (n_1 * n_2)^i$.

Induktionsschluß: Nach Induktionsannahme gilt:

$$|\{\alpha[j_q] \mid \alpha[j_q] \in F^i_{nv(\alpha)}(\alpha[j])\}| = |F^i_{nv(\alpha)}(\alpha[j])| < n_1 * (n_1 * n_2)^i .$$

Unter den Bedingungen (1) und (2) gibt es zu jedem $\alpha[j_q] \in F^i_{nv(\alpha)}(\alpha[j])$ weniger als n_2 Aktionen $\alpha[j_l]$ mit der Eigenschaft:

$$\alpha[j_q] B_a \alpha[j_l]$$

und weniger als $n_2 * (n_1 - 1)$ Aktionen mit der Eigenschaft:

$$\alpha[j_q] B_a \alpha[j_k] E_a \alpha[j_l] \;\&\; \alpha[j_k] B_a \alpha[j_l].$$

Damit gilt:

$$\begin{gathered} |F^{i+1}_{nv(\alpha)}(\alpha[j])| \le |F^i_{nv(\alpha)}(\alpha[j])| * n_2 + |F^i_{nv(\alpha)}(\alpha[j])| * n_2 * (n_1 - 1) = \\ |F^i_{nv(\alpha)}(\alpha[j])| * n_1 * n_2 < \\ n_1 * (n_1 * n_2)^{i+1} . \end{gathered}$$

und es folgt:

$$|F_{nv(\alpha)}(\alpha[j])| < \sum_{i=0}^{n_3-1} n_1 * (n_1 * n_2)^i =: n . \blacksquare$$

Mit Hilfe einer ähnlichen Beweistechnik, wie sie bereits im Abschnitt 4.2 Verwendung fand, läßt sich, orientiert an den in Abschnitt 4.3.1 veranschaulichten Problemen, der folgende Sachverhalt aufzeigen:

Satz 4.3.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem, das den folgenden vier Bedingungen genügt:

(1) $\exists n_1 \forall \alpha \forall \alpha[j] \in W(\alpha)$: [$\alpha$ aktiv in P => $(|\{i \mid \alpha[i] \in W(\alpha) \;\&\; \alpha[j] E_a \alpha[i]\}| < n_1)$];

(2) $\exists n_2 \forall \alpha_1 \alpha_2 \forall a \in A$: $[(a \in bef_P(\alpha_1)\ \&\ \alpha_1\alpha_2$ aktiv in $P\ \&$
$P_{bz(a)} \cap W(\alpha_2) = \phi) \Rightarrow |\alpha_2| < n_2]$

(3) $\exists n_3 \forall \alpha \forall \alpha[i], \alpha[j] \in W(\alpha)$ $[(\alpha$ aktiv in $P\ \&\ \alpha[i] B_a \alpha[j]) \Rightarrow$
$(|(\alpha\,|\,[t(\alpha[j]), t(\alpha[i])])| < n_3 \parallel (\exists \alpha[k] \in W(\alpha)$
$[v(\alpha[k] B_a \alpha[j])\ \&\ at(\alpha[k])\ \&\ (|(\alpha \| t(\alpha, j]), t(\alpha[k])])| < n_3|))]$

(4) $\exists n_4 \forall \alpha \forall \alpha[j], \alpha[j_1], \ldots, \alpha[j_n] \in W(\alpha)$: $[\alpha$ aktiv in $P \Rightarrow$
$[(\alpha[j] B_a \alpha[j_1] B_a \alpha[j_2] \ldots \alpha[j_{n-1}] B_a \alpha[j_n]\ \&$
$\forall i \in \{1, \ldots, n\}$: $[\neg \exists \alpha[k_i] \in W(\alpha)$: $[v(\alpha[k_i] B_a \alpha[j_i])\ \&\ at(\alpha[k_i])]]$
$\Rightarrow (|\{WP_a(\alpha[j_i])\ |\ \alpha[j_i] \in \{\alpha[j_1], \ldots, \alpha[j_n]\}\}| < n_4)]]$.

wobei $n_1, n_2, n_3, n_4 \in \mathbf{N}$.

Unter der Annahme der Gültigkeit dieser vier Bedingungen laßt sich zeigen

(*) $\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha)$: $[\alpha$ aktiv in $P \Rightarrow (|F_{nv(\alpha)}(\alpha[j])| < n)]$,

(**) $\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha)$: $[\alpha$ aktiv in $P \Rightarrow$
$(\forall \alpha[l] \in F_{nv(\alpha)}(\alpha[j])\ \ |(\alpha\,[t(\alpha[l]), t(\alpha[j])])| < n)\,|$

Beweis: Angenommen α ist aktiv in P und $\alpha[j] \in W(\alpha)$ wurde als fehlerhaft erkannt. Um zu zeigen, daß $|F_{nv(\alpha)}(\alpha[j])| < n$ (für ein von der Berechnungsfolge α unabhängiges $n \in \mathbf{N}$) gilt, genügt es nachzuweisen, daß für jedes beliebige $\alpha[l] \in F_{nv(\alpha)}(\alpha[j])$

$$|(\alpha|[t(\alpha[l]), t(\alpha[j])])| < n$$

erfüllt ist, denn für das älteste $\alpha[l] \in F_{nv(\alpha)}(\alpha[j])$ (ein ältestes $\alpha[l] \in F_{nv(\alpha)}(\alpha[j])$ muß es wegen der Endlichkeit jeder Berechnungsfolge α geben) gilt insbesondere:

$$|F_{nv(\alpha)}(\alpha[j])| \leq |(\alpha|[t(\alpha[l]), t(\alpha[j])])|.$$

Sei $\alpha[l] \in F_{nv(\alpha)}(\alpha[j])$ beliebig gewählt. Nach Definition von $F_{nv(\alpha)}(\alpha[j])$ (siehe Definition 3.2.3/3) muß gelten:

$$\exists \alpha[j_1], \ldots, \alpha[j_n] \in W(\alpha)\colon [\alpha[j] = \alpha[j_1] B_a \alpha[j_2] \ldots \alpha[j_{n-1}] B_a \alpha[j_n] = \alpha[l]\ \&$$
$$\forall i \in \{1, \ldots, n\}\ [\neg \exists \alpha[k_i]\colon v(\alpha[k_i] B_a \alpha[j_i])]],$$

und aufgrund der Beziehung (4) muß des weiteren

$$|\{WP_a(\alpha[j_i])\ |\ \alpha[j_i] \in \{\alpha[j_1], \quad, \alpha[j_n]\}\}| < n_4$$

sein.

Sei $\alpha[j_q]$ die zeitlich gesehen aktuellste Aktion der Folge α bezogen auf die Menge $\{\alpha[j_1], \ldots, \alpha[j_n]\}$, für die $\alpha[j_q] E_a \alpha[l]$ gilt. Unter den Bedingungen (1) und (2) folgt:

$$|(\alpha,[t(\alpha[l]),t(\alpha[j_q])])|<n_1{}^*n_2$$

und, falls $q>1$ (andernfalls gilt der Sonderfall $\alpha[j_q]=\alpha[j]$), mit (3):

$$|(\alpha|[t(\alpha[j_q]),t(\alpha[j_{q-1}])])|<n_3$$

sowie:

$$|\{WP_a(\alpha[j_i]) \mid \alpha[j_i]\in\{\alpha[j_1],\ldots,\alpha[j_{q-1}]\}\}|<n_4$$

aufgrund der Wahl von $\alpha[j_q]$.

Analog läßt sich, falls $q-1>1$ ist, zu $\alpha[j_{q-1}]$ wiederum ein aktuellstes $\alpha[j_r]\in\{\alpha[j_1],\ldots,\alpha[j_{q-1}]\}$ finden mit der Eigenschaft $\alpha[j_r]E_a\alpha[j_{q-1}]$, und unter den Bedingungen (1), (2) und (3) läßt sich zeigen:

$$|(\alpha|[t(\alpha[j_{q-1}]),t(\alpha[j_r])])| < n_1{}^*n_2,$$
$$|(\alpha|[t(\alpha[j_r]),t(\alpha[j_{r-1}])])| < n_3$$

sowie

$$|\{WP_a(\alpha[j_i]) \mid \alpha[j_i]\in\{\alpha[j_1],\ldots,\alpha[j_{r-1}]\}\}| < n_4-1;$$

Diese Schlußweise ist so lange möglich, bis für ein r gilt:

$$\alpha[j_r]=\alpha[j] \;\|\; \alpha[j_{r-1}]=\alpha[j]$$

bzw. schon vorher für ein q

$$\alpha[j_q]=\alpha[j] \;\|\; \alpha[j_{q-1}]=\alpha[j],$$

was nach spätestens n_4 Iterationen aufgrund der Bedingung (4) der Fall sein muß. Damit ist für ein beliebiges $\alpha[l]\in F_{mv(\alpha)}(\alpha[j])$:

$$|(\alpha|[t(\alpha[l]),t(\alpha[j])])| < (n_1{}^*n_2+n_3)^*n_4$$

nachgewiesen und mit der Definition von $n_{\ldots}:=(n_1{}^*n_2+n_3)^*n_4$ folgt die Aussage (**). Da von der Aussage (**) unmittelbar auf (*) geschlossen werden kann, ist damit Satz 4.3.2/2 bewiesen. ■

Schließlich laßt sich mit dem folgenden Satz nachweisen, daß die im Satz 4.3.2/1 als hinreichend vorgestellten Bedingungen auch notwendige Bedingungen zur Beschränkung des Umfangs einer (potentiellen) Fehlerausbreitung darstellen, wenn sie in den Punkten (1) und (2) entsprechend ergänzt werden (vgl. Satz 4.3.2/3). Die Notwendigkeit der Bedingungen (2) und (3) des Satzes 4.3.2/2 läßt sich auf ähnliche Weise nicht zeigen.

Satz 4.3.2/3: Sei $P=(A,D,w)$ ein Prozeßsystem, das den folgenden drei Bedingungen genügt:

(1) $\exists n_1 \forall \alpha \forall \alpha[j] \in W(\alpha)$: [$\alpha$ aktiv in P => ($|\{i \mid \alpha[i] \in W(\alpha)$ & $\alpha[j] E_a \alpha[i]$ & $t(\alpha[j]) > t(\alpha[i])$ & $\neg \exists \alpha[k] \in W(\alpha)$: [$v(\alpha[k] B_a \alpha[i])$ & $at(\alpha[k])\}| < n_1$)];

(2) $\exists n_2 \forall \alpha \forall \alpha[j] \in W(\alpha)$: [$\alpha$ aktiv in P => ($|\{i \mid \alpha[i] \in W(\alpha)$ & $\alpha[j] B_a \alpha[i]$ & $\neg \exists [k] \in W(\alpha)$: [$v(\alpha[k] B_a \alpha[i])$ & $at(\alpha[k])\}| < n_2$)];

(3) $\exists n_3 \forall \alpha \forall \alpha[j], \alpha[j_1], \ldots, \alpha[j_n] \in W(\alpha)$: [$\alpha$ aktiv in P [$(\alpha[j] B_a \alpha[j_1] B_a \alpha[j_2] \ldots \alpha[j_{n-1}] B_a \alpha[j_n]$ & $\forall i \in \{1,\ldots,n\}$: [$\neg \exists \alpha[k_i] \in W(\alpha)$: [$v(\alpha[k_i] B_a \alpha[j_i])$ & $at(\alpha[k_i])$]) => ($|\{WP_a(\alpha[j_i]) \mid \alpha[j_i] \in \{\alpha[j_1], \ldots, \alpha[j_n]\}\}| < n_3$)]].

wobei $n_1, n_2, n_3 \in \mathbf{N}$.

Unter der Annahme der Gültigkeit dieser drei Bedingungen läßt sich zeigen:

$$\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \iff (|F_{w(\alpha)}(\alpha[j])| < n)].$$

Beweis:

"=>": Analog zum Beweis von Satz 4.3.2/1.

"<=": Es ist offensichtlich, daß, falls es zu einem Prozeßsystem $P=(A,D,w)$ keine Konstanten $n_1, n_2, n_3 \in \mathbf{N}$ gibt, so daß für alle aktiven Berechnungen α von P und Aktionen $\alpha[j] \in W(\alpha)$ die Bedingung ((1) & (2) & (3)) gültig ist, zu P auch kein n existieren kann, so daß für alle in P aktiven Berechnungen α und $\alpha[j] \in W(\alpha)$:

$$|F_{w(\alpha)}(\alpha[j])| < n$$

gilt. ■

4.3.3. Verfeinerung des Modellierungsgrades

In den bisherigen Betrachtungen zur Fehlerausbreitung wurde die Korrektheit von Aktionen und die Fehlerausbreitung jeweils auf *Aktionenebene* modelliert. Dies bedeutet eine Vergröberung gegenüber der Realität, da die Fehlerausbreitung tatsächlich über elementarere Einheiten, nämlich die Variablen, erfolgt. Die Ausführungen dieses Abschnittes werden zeigen, wie dieser Sachverhalt im Modell berücksichtigt werden kann, ohne daß die bisher abgeleiteten Ergebnisse auch bei einer Verfeinerung des Modellierungsgrades auf *Variablenebene* ihre Gültigkeit verlieren.

Um letzteres zu demonstrieren, müssen bereits eingeführte Begriffe, wie etwa der Akzeptanztestbegriff, die basiert auf Relation B_α oder die Menge $F_{nv(\alpha)}(\alpha[j])$ der nichtverifizierten potentiellen Verursacher eines Fehlers einer Aktion $\alpha[j]$ usw., zu diesem Zweck in ihrer Bedeutung entsprechend auf Variablenebene erweitert und neue Begriffe zusätzlich definiert werden.

Definition 4.3.3/1: Sei $\alpha \subset A$ eine Aktion des Prozeßsystems $P=(A,D,w)$. Dann ist die *Zuweisungsteilrelation* ZT_α auf den im Zuweisungsteil (Operationsteil) von α vorkommenden Variablen wie folgt definiert:

$$v_i ZT_\alpha v_j \iff \exists v_i, v_j \in V_\alpha : [v_i := F_\alpha \cup N_\alpha(\ldots, v_j, \ldots)],$$

zur Definition von F_α, N_α vgl. Definition 2.1.1.2/1.

Definition 4.3.3/2: Sei $\alpha \subset A$ eine Aktion des Prozeßsystems $P=(A,D,w)$. Eine Variable v gehört zum *Bedingungsteil* von α (kurz: $v \in BT_\alpha$) genau dann, wenn:

$$v \equiv bz_{bz(\alpha)} \parallel v \text{ gehört zum Bedingungsteil } B_\alpha \text{ von } \alpha,$$

zur Definition von B_α vgl. Definition 2.1.1.2/1.

Definition 4.3.3/3: Sei $P=(A,D,w)$ ein Prozeßsystem, $AT_{i,k}$ ein Akzeptanztest auf der Variablenmenge V' und $\alpha_{i,j}$ eine den Akzeptanztest ausführende Aktion. Dann ist $\alpha_{i,j}$ in der Lage den Wert der Variablen v zu *verifizieren/validieren* genau dann, wenn $v \in V'$.

Kurzschreibweise $AT_{\alpha_{i,j}}(v)=\textbf{true} \iff$ die akzeptanztestausführende Aktion $\alpha_{i,j}$ ist in der Lage den Wert der Variablen v zu verifizieren/validieren.

Definition 4.3.3/4: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnung von P. Eine Aktion $\alpha[i] \in W(\alpha)$ ***akzeptiert*** (*verifiziert*, *validiert*) die Aktion $\alpha[j]$ bzgl. deren Modifikation der Variablen $v \in V_P$, kurz: $\alpha[i] A_{\alpha,[v]} \alpha[j]$, genau dann, wenn:

$$\begin{aligned}&\alpha[i] \text{ ist positiver (erfolgreich ausgeführter) Akzeptanztest } \&\\ &AT_{\alpha[i]}(v)=\textbf{true} \;\&\; v \in SV_{\alpha[j]} \;\&\; t(\alpha[i]) > t(\alpha[j]) \;\&\\ &\neg \exists \alpha[k] \in W(\alpha): [v \in SV_{\alpha[k]} \;\&\; t(\alpha[k]) \in]t(\alpha[j]), t(\alpha[i])[].\end{aligned}$$

Definition 4.3.3/5: Sei $P=(A,D,w)$ ein Prozeßsystem und α ein aktiver Ablauf von P. Eine Aktion $\alpha[i]$ ***akzeptiert*** (*lehnt*) die Aktion $\alpha[j]$ bzgl. deren Modifikation der Variablen v *nicht* (*ab*), kurz: $\alpha[i] N_{\alpha,[v]} \alpha[j]$, genau dann, wenn:

$$\begin{aligned}&\alpha[i] \text{ ist negativer (zur Fehlererkennung führender) Akzeptanztest } \&\\ &AT_{\alpha[i]}(v)=\textbf{false} \;\&\; v \in SV_{\alpha[j]} \;\&\; t(\alpha[i]) > t(\alpha[j]) \;\&\\ &\neg \exists \alpha[k] \in W(\alpha): [v \in SV_{\alpha[k]} \;\&\; t(\alpha[k]) \in]t(\alpha[j]), t(\alpha[i])[] \;\&\end{aligned}$$

$$\neg\exists\alpha[l]\in W(\alpha):\ [\alpha[l]A_{a,[v]}\alpha[j]].$$

Definition 4.3.3/6: Sei $P=(A,D,w)$ ein Prozeßsystem, α eine aktive Berechnungsfolge. Dann läßt sich die *basiert auf Relation auf Variablenebene*, kurz $B_{a,[v]}$, wie folgt definieren:

$$\alpha[j]B_{a,[v]}\alpha[i] \ ::<=>\ (\alpha[j]B_a\alpha[i]\ \&\ v\in LV_{\alpha[j]}\cap SV_{\alpha[i]}\ \&\ \neg\exists\alpha[k]\in W(\alpha):\ [t(\alpha[j])>t(\alpha[k])>t(\alpha[i])\ \&\ v\in SV_{\alpha[k]}]).$$

Definition 4.3.3/7: Die *Menge aller potentiellen (nichtverifizierten) Verursacher des Fehlverhaltens einer Aktion* $\alpha[i]\in W(\alpha)$ kann unter Berücksichtigung der vorangehenden Definitionen dieses Abschnittes im Vergleich zur Definition 3.2.3/3 auf folgende Weise präziser beschrieben werden:

$$\begin{aligned}F_{w(\alpha)}(\alpha[i]) \ ::=\ \{&\alpha[j]\mid \exists\alpha[j_1],\ldots,\alpha[j_n]\in W(\alpha)\exists v_1,\ldots,v_n\in V_P:\\ &[\alpha[i]B_{a,[v_1]}\alpha[j_1]B_{a,[v_2]}\alpha[j_2]\ldots\alpha[j_{n-1}]B_{a,[v_n]}\alpha[j_n]=\alpha[j]\ \&\\ &\exists v\in V_{\alpha[i]}\neg\exists\alpha[k]\in W(\alpha):\ [\alpha[k]A_{a,[v]}\alpha[i]\ \&\\ &(vZT_{\alpha[i]}v_1\parallel v_1\in BT_{\alpha[i]})\ \&\\ &\neg\exists\alpha[k_1]\in W(\alpha)\ [\alpha[k_1]A_{a,[v_1]}\alpha[j_1]]\\ &\forall l\in\{1,\ldots,n-1\}:\ [\neg\exists\alpha[k_{l+1}]\in W(\alpha):\ [\alpha[k_{l+1}]A_{a,[v_{l+1}]}\alpha[j_{l+1}]\ \&\\ &\qquad(v_lZT_{\alpha[j_l]}v_{l+1}\parallel v_{l+1}\in BT_{\alpha[j_l]})]]]\}\cup\{\alpha[i]\}.\end{aligned}$$

Unter Verwendung der neu eingeführten Begriffe lassen sich die Analoga zu den Sätzen 4.3.2/1 bis 3 wie folgt formulieren; die zugehörigen Beweise behalten praktisch in unveränderter Form ihre Gültigkeit bei.

Satz 4.3.3/1: Sei $P=(A,D,w)$ ein Prozeßsystem, das den folgenden drei Bedingungen genügt:

(1) $\exists n_1\forall\alpha\forall\alpha[j]\in W(\alpha):\ [\alpha$ aktiv in $P\ =>\ (|\{i\mid\alpha[i]\in W(\alpha)\ \&\ \alpha[j]E_a\alpha[i]\}|<n_1)]$;

(2) $\exists n_2\forall\alpha\forall\alpha[j]\in W(\alpha):\ [\alpha$ aktiv in $P\ =>\ (|\{i\mid\alpha[i]\in W(\alpha)\ \&\ \alpha[j]B_a\alpha[i]\}|<n_2)]$;

(3) $\exists n_3\forall\alpha\forall\alpha[j],\alpha[j_1],\ldots,\alpha[j_n]\in W(\alpha)\forall v_1,\ldots,v_n\in V_P:\ [\alpha$ aktiv in $P\ =>$
$[(\alpha[j]B_{a,[v_1]}\alpha[j_1]B_{a,[v_2]}\alpha[j_2]\ldots\alpha[j_{n-1}]B_{a,[v_n]}\alpha[j_n]\ \&$
$\forall i\in\{1,\ldots,n\}:\ [\neg\exists\alpha[k_i]\in W(\alpha):\ [\alpha[k_i]A_{a,[v_i]}\alpha[j_i]])$
$=>(|\{WP_a(\alpha[j_i])\mid\alpha[j_i]\in\{\alpha[j_1],\ldots,\alpha[j_n]\}\}|<n_3)]]$.

wobei $n_1,n_2,n_3\in\mathbf{N}$.

Unter der Annahme der Gültigkeit dieser drei Bedingungen läßt sich zeigen:

$$\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \Rightarrow (|F_{nv(\alpha)}(\alpha[j])| < n)]$$

Beweis: Analog zum Beweis von Satz 4.3.2/1. ■

Satz 4.3.3/2: Sei $P=(A,D,w)$ ein Prozeßsystem, das den folgenden vier Bedingungen genügt

(1) $\exists n_1 \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \Rightarrow (|\{i \mid \alpha[i] \in W(\alpha) \ \& \ \alpha[j] E_a \alpha[i]\}| < n_1)]$;

(2) $\exists n_2 \forall \alpha_1 \alpha_2 \forall a \in A: [(a \in bef_P(\alpha_1) \ \& \ \alpha_1 \alpha_2 \text{ aktiv in } P \ \& \ P_{bz(a)} \cap W(\alpha_2) = \phi) \Rightarrow |\alpha_2| < n_2]$

(3) $\exists n_3 \forall \alpha \forall \alpha[i], \alpha[j] \in W(\alpha): [(\alpha \text{ aktiv in } P \ \& \ \alpha[i] B_a \alpha[j]) \Rightarrow (|\{\alpha \mid [t(\alpha[j]), t(\alpha[i])]\}| < n_3 \ \| \ (\exists \alpha[k] \in W(\alpha) \forall v \in SV_{\alpha[j]}: [\alpha[k] A_{a,[v]} \alpha[j] \ \& \ |(\alpha[[t(\alpha[j]), t(\alpha[k])]]| < n_3)])]$

(4) $\exists n_4 \forall \alpha \forall \alpha[j], \alpha[j_1], \ldots, \alpha[j_n] \in W(\alpha) \forall v_1, \ldots, v_n \in V_P: [\alpha \text{ aktiv in } P \Rightarrow [(\alpha[j] B_{a,[v_1]} \alpha[j_1] B_{a,[v_2]} \alpha[j_2] \ldots \alpha[j_{n-1}] B_{a,[v_n]} \alpha[j_n] \ \& \ \forall i \in \{1, \ldots, n\}: \neg \exists \alpha[k_i] \in W(\alpha): [\alpha[k_i] A_{a,[v_i]} \alpha[j_i]]) \Rightarrow (|\{WP_a(\alpha[j_i]) \mid \alpha[j_i] \in \{\alpha[j_1], \ldots, \alpha[j_n]\}\}| < n_4)]]$.

wobei $n_1, n_2, n_3, n_4 \in \mathbf{N}$.

Unter der Annahme der Gültigkeit dieser vier Bedingungen läßt sich zeigen:

(*) $\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \Rightarrow (|F_{nv(\alpha)}(\alpha[j])| < n)]$,

(**) $\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \Rightarrow (\forall \alpha[l] \in F_{nv(\alpha)}(\alpha[j]): |(\alpha[[t(\alpha[l]), t(\alpha[j])]]| < n)]$.

Beweis: Analog zum Beweis von Satz 4.3.2/2. ■

Satz 4.3.3/3: Sei $P=(A,D,w)$ ein Prozeßsystem, das den folgenden drei Bedingungen genügt:

(1) $\exists n_1 \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \Rightarrow (|\{i \mid \alpha[i] \in W(\alpha) \ \& \ \alpha[j] E_a \alpha[i] \ \& \ t(\alpha[j]) > t(\alpha[i]) \ \& \ \neg \exists \alpha[k] \in W(\alpha) \forall v \in SV_{\alpha[i]}: [\alpha[k] A_{a,[v]} \alpha[i]]\}| < n_1)]$;

(2) $\exists n_2 \forall \alpha \forall \alpha[j] \in W(\alpha): [\alpha \text{ aktiv in } P \Rightarrow (|\{i \mid \alpha[i] \in W(\alpha) \ \& \ \alpha[j] B_a \alpha[i] \ \& \ \neg \exists \alpha[k] \in W(\alpha) \forall v \in SV_{\alpha[i]}: [\alpha[k] A_{a,[v]} \alpha[i]]\}| < n_2)]$.

(3) $\exists n_3 \forall \alpha \forall \alpha[j], \alpha[j_1], \ldots, \alpha[j_n] \in W(\alpha) \forall v_1, \ldots, v_n \in V_P$: [$\alpha$ aktiv in P =>
$[(\alpha[j] B_{a,[v_1]} \alpha[j_1] B_{a,[v_2]} \alpha[j_2] \ldots \alpha[j_{n-1}] B_{a,[v_n]} \alpha[j_n]$ &
$\forall i \in \{1, \ldots, n\}$: [$\neg \exists \alpha[k_i] \in W(\alpha)$: $[\alpha[k_i] A_{a,[v_i]} \alpha[j_i]]$)
=> $(|\{ WP_a(\alpha[j_i]) \mid \alpha[j_i] \in \{\alpha[j_1], \ldots, \alpha[j_n]\}\}| < n_3)]]$

wobei $n_1, n_2, n_3 \in \mathbf{N}$.

Unter der Annahme der Gültigkeit dieser drei Bedingungen läßt sich zeigen:

$\exists n \in \mathbf{N} \forall \alpha \forall \alpha[j] \in W(\alpha)$: [$\alpha$ aktiv in P <=> $(|F_{w(\alpha)}(\alpha[j])| < n)$].

Beweis: Analog zum Beweis von Satz 4.3.2/3. ■

4.4. Kriterien für Dominoeffektfreiheit

In den Abschnitten 4.2 und 4.3 wurden *notwendige und hinreichende* Bedingungen für das Nichtauftreten des Dominoeffektes sowohl für den Fall der Berücksichtigung als auch den der Nichtberücksichtigung potentieller Fehlerverursacher abgeleitet. Zusammenfassend läßt sich festhalten, daß unter der Annahme,

> zwischen dem Ausführungszeitpunkt $t(\alpha[i])$ einer als fehlerhaft erkannten oder durch einen Rechensystemfehler unterbrochenen Aktion $\alpha[i]$ und dem Fehlererkennungszeitpunkt, d.h. dem Zeitpunkt der Einleitung von Fehlerbehandlungsmaßnahmen, dürfen nur beschränkbar viele Aktionen ausgeführt worden sein,

bei Nichtberücksichtigung (potentieller) Fehlerverursacher der Dominoeffekt bzgl. aktiver Berechnungen eines Prozeßsystems $P=(A,D,w)$ dann nicht auftreten kann, wenn die folgenden Kriterien erfüllt sind:

(1) der Umfang der *zyklischen Zurücksetzung* im Fehlerfall, bezogen auf jeden Prozeß von P, durch eine Konstante berechnungsunabhängig beschränkbar ist,

(2) die *Zahl der Aktionen einer jeden Recoveryeinheit* einer aktiven Berechnung α von P berechnungsunabhängig durch eine Konstante beschränkt werden kann,

(3) die *Länge jeder Prozeßinaktivphase* eines jeden Prozesses von P innerhalb einer aktiven Berechnung von P berechnungsunabhängig durch eine Konstante beschränkt werden kann,

und bei Berücksichtigung (potentieller) Fehlerverursacher der Dominoeffekt dann nicht auftreten kann, wenn zusätzlich die folgenden Bedingungen erfüllt sind:

(4) die *Zahl der (direkten) nichtverifizierten Abhängigkeitsbeziehungen* einer jeden Aktion in aktiven Berechnungen α von P durch eine Konstante berechnungsunabhängig beschränkt werden kann,

(5) die *Zahl der Recoveryeinheiten, über die sich ein Fehler in aktiven Berechnungen* α von P *(unentdeckt) fortpflanzen kann*, durch eine Konstante berechnungsunabhängig beschränkt werden kann,

(6) die *Zahl der Berechnungsschritte von der Änderung eines jeden Variablenwertes bis zu seiner Verifikation* in aktiven Berechnungen α von P durch eine Konstante

berechnungsunabhängig beschränkt werden kann

Die Bedingungen (1) bis (6) stellen hinreichende Bedingungen für die Dominoeffektfreiheit eines Prozeßsystems dar. Hinreichende und gleichzeitig notwendige Bedingungen können den Sätzen 4.2.4/2 und 4.3.2/3 entnommen werden.

Sämtliche Ergebnisse wurden unter Zugrundelegung des Prozeßsystembegriffs, wie er in Definition 2.1.1.2/1 präzisiert wurde, abgeleitet, und gelten deshalb sowohl für den Spezialfall der Synchronisationssysteme und der Nachrichtensysteme als auch für den Fall von Prozeßsystemen der Klassen unabhängiger, kooperierender, kommunizierender und koordinierender Prozesse.
Des weiteren behalten die Ergebnisse, obwohl bei ihrer Herleitung implizit, dies drückt sich in der Definition der Menge der zurückzusetzenden Aktionen $Z_\alpha(\bigcup_{\alpha[i]\in V(\alpha)\wedge f_\alpha(\alpha[i])} F_{rv(\alpha)}(\alpha[i]))$ einer Berechnungsfolge α aus, die Fehlerbehandlungsfunktion r_1 einbezogen wurde, ihre Gültigkeit auch im Zusammenhang mit den Recoveryfunktionen r_2 und/oder r_3 bei, vorausgesetzt, diese sind in einer angenommenen Fehlersituation anwendbar, denn im Falle ihrer Nutzbarkeit verursachen r_2 und r_3, wie im Abschnitt 3.4 dargelegt wurde, im Vergleich zu r_1 höchstens ebensoviel, in der Regel jedoch weniger, Recoveryaufwand.

Auch die folgenden Untersuchungen orientieren sich am allgemeinen Prozeßsystembegriff und der Definition der Recoveryfunktion r_1. Sie sind daher ohne weiteres auf die oben erwähnten unterschiedlichen Typen von Prozeßsystemen und Fehlerbehandlungsfunktionen, die Spezialfälle darstellen, übertragbar.
Die Betrachtungen beziehen sich auf die Situation der Nichtberücksichtigung (potentieller) Fehlerverursacher. Das angestrebte Ziel ist die Identifizierung von Berechnungseigenschaften, welche Bedingung (1) für das Nichtauftreten des Dominoeffektes, d.h. den Umfang der zyklischen Zurücksetzung im Fehlerfall, beschränkbar machen und damit die wesentliche Voraussetzung für die Dominoeffektfreiheit in der betrachteten Situation erfüllen.
Der Fall der Berücksichtigung (potentieller) Fehlerverursacher bzw. der Fehlerausbreitung, für den es von Interesse wäre Bedingungen für die Beschränkbarkeit der Fehlerfortpflanzung zu identifizieren, wird in dieser Arbeit nicht weiter verfolgt. Prinzipiell würde in diesem Bereich die Aufgabe darin bestehen herauszufinden, wie Akzeptanztests innerhalb einer Berechnung anzuordnen sind, und welche Eigenschaften sie haben müssen, damit eine Fehlerfortpflanzung nur über beschränkbar viele Berechnungsschritte möglich ist. Die genaue Klärung dieser beiden Fragen bleibt zukünftigen Aktivitäten auf dem Gebiet der Fehlerbehandlung bzw. den Überlegungen des Lesers vorbehalten.

5. Beispiele dominoeffektfreier Prozeßsystemklassen

Einen wesentlichen Schlüssel zur Konstruktion dominoeffektfreier Prozeßsysteme bzw. zur Verhinderung des Dominoeffekts bildet die Begrenzung der zyklischen Zurücksetzung. Häufig wird in der Literatur, wie etwa in [Kan81], mit dem Problem der unbeschränkbaren zyklischen Zurücksetzung intuitiv auch das Phänomen Dominoeffekt umschrieben.
Da sich die folgenden Ausführungen nur auf den Fall der Nichtberücksichtigung (potentieller) Fehlerverursacher konzentrieren und auf den Fall der Berücksichtigung potentieller Fehlerverursacher nur am Rande (in Abschnitt 5.6) eingehen, bedeutet die Lösung dieses Problems innerhalb eines Prozeßsystems $P=(A,D,w)$, mit geringen, leicht zu erfüllenden Zusätzen, auch gleichzeitig, wie noch zu sehen sein wird, die Dominoeffektfreiheit von P. Aus diesem Grunde werden im weiteren Bedingungen abgeleitet, unter welchen das Problem der zyklischen Zurücksetzung, wie es im Abschnitt 4.2 definiert wurde, nicht auftreten kann. Es wird sich zeigen, daß diesem Phänomen u.a. durch sorgfältiges Auswählen und Anordnen von Wiederaufsetzpunkten innerhalb einer Berechnung beizukommen ist. Hier lassen sich insbesondere zwei Strategien zum Setzen von Wiederaufsetzpunkten unterscheiden. Bei vom Programmierer innerhalb der Prozesse eines Prozeßsystems eingefügten Wiederaufsetzpunkten, die im Sinne von Abschnitt 2.2.2.1 als prozeßsysteminitiiert bezeichnet werden können, läßt sich, in Anlehnung an die Bezeichnungsweisen beim Deadlockproblem (vgl. [Hof84]), von einer (*Dominoeffekt-*) *Vermeidungsstrategie*, bei vom Betriebssystem gesetzten, in Abschnitt 2.2.2.1 als betriebssysteminitiiert bezeichneten Wiederaufsetzpunkten kann man von einer (*Dominoeffekt-*) *Verhinderungsstrategie*, die auf der *Erkennung* von Situationen, welche zum Dominoeffekt führen können, basiert, sprechen.

Die Realisierung der (Dominoeffekt-) Verhinderungs- bzw. Vermeidungsstrategien findet ihren Niederschlag in wiederaufsetzpunktgesicherten globalen Leseaufrufen, prozeßwechselbezogen gesicherten, struktur-/abhängigkeitsorientiert gesicherten und transaktionsorientiert gesicherten Prozeßbeziehungen, kann aber auch in der Sicherstellung rein azyklischer Prozeßbeziehungen bestehen. Näheres zu diesen Punkten enthalten die nachfolgenden Abschnitte dieses Kapitels.

Obwohl die nachstehenden Ergebnisse auch für den Fall von Systemen unabhängiger Prozesse Gültigkeit besitzen, sei an dieser Stelle darauf hingewiesen, daß, da in derartigen Systemen, aufgrund der in Abschnitt 2.1.4.1 präzisierten Eigenschaft der Unabhängigkeit, das Problem der zyklischen Zurücksetzung nicht auftreten kann, es zur Vermeidung des Dominoeffektes in Systemen unabhängiger Prozesse ausreichend ist, für rechtzeitige Fehlererkennung und eine berechnungsunabhängig beschränkbare Recoveryeinheitsgröße zu sorgen.

5.1. Wiederaufsetzpunktgesicherte globale Leseaufrufe

Die im weiteren Verlauf dieses Abschnitts zusammengestellten Sätze werden zeigen, daß das Problem der unbeschränkbaren zyklischen Zurücksetzung u.a. dann nicht auftreten kann, wenn die globalen Leseaufrufe innerhalb einer Berechnung in der Weise gesichert werden, daß zwischen ihnen und vorausgehenden globalen Schreibaufrufen ein Wiederaufsetzpunkt besteht. Dieser Sachverhalt und Varianten davon, die in diesem Fall auch die Dominoeffektfreiheit von Prozeßsystemen auf einfache Weise nachzuweisen gestatten, werden im folgenden bewiesen. Zuvor jedoch werden zwei in diesem Zusammenhang benötigte Hilfssätze abgeleitet.

5.1.1. Charakterisierung vermeidbarer Zurücksetzung

In den beiden folgenden Hilfssätzen werden Beziehungen aufgezeigt, welche zwischen Aktionen einer Berechnungsfolge bestehen müssen, damit bei einer Fehlerbehandlung vermeidbare Zurücksetzung überhaupt auftreten kann. In Abschnitt 5.1.2 werden dann darauf aufbauend Bedingungen identifiziert, unter welchen die Zahl der vermeidbar zurückzusetzenden Aktionen einer Berechnung beschränkt werden kann. Da nach der in Abschnitt 4.2 eingeführten Charakterisierung des Problems der unbeschränkbaren zyklischen Zurücksetzung dieses prinzipiell nur dann auftreten kann, wenn die Zahl der vermeidbar zurückzusetzenden Aktionen nicht durch eine Konstante berechnungsunabhängig beschränkbar ist, werden in Abschnitt 5.1.2 damit auch Bedingungen abgegrenzt, unter welchen das Problem der unbeschränkbaren zyklischen Zurücksetzung nicht vorkommen kann. Die beiden für das weitere benötigten Hilfssätze lauten:

Hilfssatz 5.1.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1, P_2, \ldots, P_n$. Wird Prozeß P_i, $1 \le i \le n$, im Zustand $val_P(\alpha)$ zum Wiederaufsetzpunkt $WP_a(\alpha[l])$, $\alpha[l] \in W(\alpha)$, zurückgesetzt, so läßt sich die älteste Aktion $\alpha[p_j]$ des Prozesses P_j, $1 \le j \le n$, deren Zurücksetzung bei diesem Vorgang nicht vermieden werden kann, für $j \neq i$ wie folgt charakterisieren: $\alpha[p_j]$ muß eine Aktion sein, die mehreren Prozessen (mindestens zwei Prozessen) gemeinsame Variablen liest, d.h. $gL(\alpha[p_j])=$**true**, und es muß ein $\alpha[k] \in W(\alpha)$ existieren, so daß $\alpha[p_j] B_a \alpha[k]$ & $\neg proc_P(\alpha[p_j],\alpha[k])$ gilt.

Beweis: Da die Zurücksetzung von $\alpha[p_j]$ nach Voraussetzung unvermeidbar ist, muß $\alpha[p_j] B_a^* \alpha[p_i]$ gelten. Insbesondere muß eine Aktion $\alpha[k]$ existieren, so daß gilt: $\alpha[p_j] B_a \alpha[k] B_a^* \alpha[p_i]$.
Da $\alpha[p_j]$ die älteste Aktion des Prozesses P_j ist, deren Zurücksetzung unvermeidbar ist, und da wegen $\alpha[p_j] B_a \alpha[k]$ $t(\alpha[p_j]) > t(\alpha[k])$ gilt, können die Aktionen $\alpha[p_j]$ und $\alpha[k]$ nicht dem gleichen Prozeß angehören (sonst wäre $\alpha[k]$ die älteste Aktion von P_j mit der genannten Eigenschaft im Widerspruch zur Voraussetzung).
Aufgrund der Definition von B_a (siehe 2.1.3.1/1), der Beziehung $\alpha[p_j] B_a \alpha[k]$ und

$\neg proc_P(\alpha[p_j],\alpha[k])$ läßt sich folgern:

$$\{LV_{a[p_j]} - \{lV_{a[p_j]}\}\} \cap \{SV_{a[k]} - \{lV_{a[k]}\}\} \neq \phi,$$

d.h. $\alpha[p_j]$ ist eine Aktion, die mehreren Prozessen gemeinsame Variablen liest, und damit ist $gL(\alpha[p_j])$=**true**. ■

Hilfssatz 5.1.1/2: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1, P_2, \ldots, P_n$. Sei $\alpha[i]$ eine Aktion, deren Zurücksetzung im Rahmen einer Fehlerbehandlung im Prozeßsystem-Zustand $val_P(\alpha)$ unvermeidbar ist, und sei $\alpha[p_j]$ die älteste Aktion des Prozesses P_j $1 \leq j \leq n$, deren Zurücksetzung dabei nicht vermieden werden kann. $\alpha[k]$ sei eine Aktion, für die $\alpha[k](B_a \cup E_a)\alpha[i]$ gilt.
Unter diesen Voraussetzungen ist die Zurücksetzung von $\alpha[k]$ im Rahmen der genannten Fehlerbehandlung genau dann vermeidbar, wenn gilt:

$$t(\alpha[k]) < t(\alpha[p_j]) \leq t(\alpha[i]) \quad \& \quad \alpha[k]E_a\alpha[p_j]E_a\alpha[i].$$

Beweis "=>": Ist die Zurücksetzung von $\alpha[k]$ vermeidbar, so kann $\alpha[k]B_a\alpha[i]$ nicht gelten, denn, da die Zurücksetzung von $\alpha[i]$ unvermeidbar ist, würde dies andernfalls bedeuten, daß auch die Zurücksetzung von $\alpha[k]$ nicht vermieden werden kann. Da aber nach Voraussetzung zwischen $\alpha[k]$ und $\alpha[i]$ die Beziehung $\alpha[k](B_a \cup E_a)\alpha[i]$ besteht, bedeutet dies, daß zwischen $\alpha[k]$ und $\alpha[i]$ die Beziehungen $\alpha[k]E_a\alpha[i]$ und $t(\alpha[k])<t(\alpha[i])$ bestehen müssen. Des weiteren muß, da die Zurücksetzung von $\alpha[i]$ nach Voraussetzung unvermeidbar ist, ein $\alpha[p_j]$ existieren, für das gilt: $t(\alpha[p_j])\leq t(\alpha[i])$ & $proc_P(\alpha[p_j],\alpha[i])$. Mit diesem Zwischenergebnis folgt aufgrund der vorab nachgewiesenen Beziehungen $\alpha[k]E_a\alpha[i]$ und $t(\alpha[k])<t(\alpha[i])$:

$$t(\alpha[k])<t(\alpha[p_j]) \quad \& \quad \alpha[k]E_a\alpha[p_j],$$

und aufgrund der Reflexivität/Transitivität der Relation E_a kann auf $\alpha[p_j]E_a\alpha[i]$ geschlossen werden. Damit sind die Beziehungen $t(\alpha[k])<t(\alpha[p_j])<t(\alpha[i])$ und $\alpha[k]E_a\alpha[p_j]E_a\alpha[i]$ nachgewiesen.

"<=": Ist $\alpha[k]E_a\alpha[p_j]$ erfüllt, so muß $\alpha[k]$ bei Zurücksetzung von $\alpha[p_j]$ ebenfalls zurückgesetzt werden. Gilt zusätzlich $t(\alpha[k])<t(\alpha[p_j])$, wobei $\alpha[p_j]$ die älteste Aktion des Prozesses P_j ist, deren Zurücksetzung unvermeidbar ist, muß die Zurücksetzung von $\alpha[k]$ vermeidbar sein, denn da $\alpha[k]E_a\alpha[p_j]$ und damit $proc_P(\alpha[k],\alpha[p_j])$ erfüllt ist, wäre $\alpha[p_j]$ sonst nicht die älteste Aktion von P_j mit der genannten Eigenschaft. ■

5.1.2. Wirksamkeit globaler Leseaufrufsicherung

Vor dem Hintergrund der Charakterisierung vermeidbarer Zurücksetzung in den Hilfssatzen 5.1.1/1 und 2 lassen sich die folgenden drei Sätze ableiten, welche die Wirksamkeit der Sicherung globaler Leseaufrufe durch Wiederaufsetzpunkte nachweisen und damit aufzeigen, wie man sich vor dem Problem der nicht beschränkbaren

zyklischen Zurücksetzung schützen und dominoeffektfreie Prozeßsysteme konstruieren kann. Parallelen zu den Ergebnissen der Arbeiten von D.L. Russell [Rus78], [Rus80], und K. Kant, [Kan81], werden im Kapitel sechs, das sich mit Ergebnissen themaverwandter Arbeiten beschäftigt, hergestellt. An dieser Stelle sei nur erwähnt, daß sowohl die im vorangehenden Abschnitt bewiesenen Hilfssätze als auch die nachfolgend abgeleiteten Sätze 5.1.2/1 und 5.1.2/2 Verallgemeinerungen von in [Rus80] für Erzeuger-/Verbrauchersysteme nachgewiesener Aussagen darstellen.

In Satz 5.1.2/1 wird zunächst gezeigt, daß in einem Prozeßsystem vermeidbare Zurücksetzung dann nicht auftreten kann, wenn unmittelbar vor jeder globalen Leseaktion ein Wiederaufsetzpunkt angelegt wird.
Mittels der Sätze 5.1.2/2 und 3 wird dann nachgewiesen, daß die Voraussetzungen, unter denen die Aussage von Satz 5.1.2/1 abgeleitet werden kann, weiter abgeschwächt werden können, ohne daß sich die Grundaussage von Satz 5.1.2/1, nämlich die Beschränkbarkeit der Menge der im Fehlerfall vermeidbar zurückzusetzenden Aktionen, ändert.
Eine Situation, in welcher die Aussagen der Sätze 5.1.2/1, 2 und 3 in Verbindung mit den in Kapitel 4 abgeleiteten Kriterien zum Nachweis der Dominoeffektfreiheit eines Prozeßsystems ausreichen, skizziert Satz 5.1.2/4.

Satz 5.1.2/1 Sei $P=(A,D,w)$ ein Prozeßsystem und unmittelbar vor jeder Aktion $\alpha\in A$, die mehreren Prozessen gemeinsame Variablen liest, d.h. für die $gL(\alpha)=\mathbf{true}$ gilt, sei in jeder Berechnung α bzgl. des Prozesses $P_{bz(\alpha)}$ ein Wiederaufsetzpunkt gesetzt worden. Dann ist die Menge der zurückzusetzenden Aktionen, deren Zurücksetzung vermeidbar wäre, leer, kurz:

$$V_{Z_\alpha}\Big(\bigcup_{\alpha[i]\in W(\alpha)\,\&\,f_\alpha(\alpha[i])} F_{nw(\alpha)}(\alpha[i])\Big)=\phi.$$

Beweis: Angenommen der Prozeß P_i des Prozeßsystems P muß im Zustand $val_P(\alpha)$ zum Wiederaufsetzpunkt $WP_\alpha(\alpha[q])$ zurückgesetzt werden und $\alpha[p_i]$ ist die älteste Aktion von P_i mit der Eigenschaft: $t(WP_\alpha(\alpha[q]))\le t(\alpha[p_i])$ (man beachte, daß die Zurücksetzung von $\alpha[p_i]$ wegen $proc_P(\alpha[q],\alpha[p_i])$ und $t(WP_\alpha(\alpha[q]))\le t(\alpha[p_i])$ unvermeidbar ist). Dann muß für jede Aktion $\alpha[k]$, die bei dem zugehörigen Recoveryvorgang zurückgesetzt werden muß, gelten:

$$\alpha[k](B_\alpha\cup E_\alpha)^*\alpha[p_i].$$

Aufgrund des Hilfssatzes 3.2.4/5 muß es endlich viele Aktionen $\alpha[k_1],\ldots,\alpha[k_m]$ geben, so daß gilt:

$$\alpha[k]=\alpha[k_m](B_\alpha\cup E_\alpha)\alpha[k_{m-1}](B_\alpha\cup E\alpha)\ldots\alpha[k_1](B_\alpha\cup E_\alpha)\alpha[p_i].$$

Da $\alpha[p_i]$ die älteste Aktion des Prozesses P_i mit der Eigenschaft $t(WP_a(\alpha[q]))\leq t(\alpha[p_i])$ ist, kann auf $\alpha[k_1]B_a\alpha[p_i]$ $\parallel$ $(proc_P(\alpha[k_1],\alpha[p_i])$ & $t(\alpha[p_i])\leq t(\alpha[k_1]))$ geschlossen werden, und damit ist die Zurücksetzung von $\alpha[k_1]$ unvermeidbar.
Betrachtet man nun die Beziehung $\alpha[k_l](B_a\cup E_a)\alpha[k_{l-1}]$ unter der Annahme die *Zurücksetzung von* $\alpha[k_{l-1}]$ *sei nicht zu vermeiden*, und wird mit $\alpha[p_j]$ die älteste Aktion des Prozesses P_j bezeichnet, deren Zurücksetzung unvermeidbar ist, so kann nach Hilfssatz 5.1.1/2 die Zurücksetzung von $\alpha[k_l]$ nur dann vermieden werden, wenn $t(\alpha[k_l])<t(\alpha[p_j])$ & $\alpha[k_l]E_a\alpha[p_j]$ und damit $proc_P(\alpha[k_l],\alpha[p_j])$ für irgendeinen Prozeß P_j von P gilt, $1\leq j\leq n$ (unter der Annahme P bestehe aus den n Prozessen $P_1, \ldots, P_n$).
Durch vollständige Fallunterscheidung wird nun gezeigt, daß es ein solches $\alpha[p_j]$ nicht geben kann.

Fall 1 $(i=j)$: Ist $i=j$, dann kann, da $\alpha[p_i]$ die älteste Aktion mit der Eigenschaft

$$t(\alpha[p_i])\geq t(WP_a(\alpha[q]))\ \&\ proc_P(\alpha[q],\alpha[p_i])$$

ist, $\alpha[k_l]E_a\alpha[p_i]$ und gleichzeitig $t(\alpha[k_l])<t(\alpha[p_i])$ nicht gelten.

Fall 2 $(i\neq j)$: Ist $i\neq j$, dann muß aufgrund des Hilfssatzes 5.1.1/1 $gL(\alpha[p_j])$=**true** gelten, und da sich nach Voraussetzung des Satzes unmittelbar vor jeder Aktion $\alpha\in A$, für die $gL(\alpha)$=**true** gilt, in $P_{bz(\alpha)}$ ein Wiederaufsetzpunkt befindet, kann es auch in diesem Fall zu $\alpha[k_l]$ kein $\alpha[p_j]$ mit der Eigenschaft $\alpha[k_i]E_a\alpha[p_j]$ & $t(\alpha[k_l])<t(\alpha[p_j])$ geben. Insgesamt heißt das, daß die Zurücksetzung von $\alpha[k_l]$ (genauso wie die von $\alpha[k_{l-1}]$) unvermeidbar ist.
Mit Hilfe vollständiger Induktion läßt sich auf diese Weise zeigen, daß auch die Zurücksetzung von $\alpha[k]=\alpha[k_m]$ unvermeidbar ist. Damit ist nachgewiesen, wie vorangehend dargelegt, daß bei jedem beliebigen Recoveryvorgang im Prozeßsystem P nur Aktionen zurückgesetzt werden, deren Zurücksetzung nicht vermieden werden kann ■

Satz 5.1.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1, \ldots, P_n$, in dem für jeden aktiven Prozeßablauf α in bezug auf jeden Prozeß P_i, $1\leq i\leq n$, gilt:

$$(*)\ \forall\alpha[j],\alpha[k]\in W(\alpha):\ [(t(\alpha[j])<t(\alpha[k])\ \&\ proc_P(\alpha[j],\alpha[k])\ \&\ gS(\alpha[j])\ \&\ gL(\alpha[k]))$$
$$\Rightarrow\ (WP_a(\alpha[j])\neq WP_a(\alpha[k]))].$$

Des weiteren existiere eine *maximale Recoveryeinheitsgröße* $g_{re}\in\mathbf{N}$, so daß für alle aktiven Berechnungen α von P gilt:

$$g_{re_\alpha} ::= \max_{\alpha[k]\in W(\alpha)}|\{\alpha[j]\mid\alpha[j]\in W(\alpha)\ \&\ \alpha[j]E_a\alpha[k]\}| < g_{re}.$$

Dann ist die Anzahl der beim Zurücksetzen eines Prozesses P_i von P zum Wiederaufsetzpunkt $WP_a(\alpha[k])$ im Zustand $val_P(\alpha)$ zurückzusetzenden Aktionen, deren Zurücksetzung vermeidbar wäre, kleiner oder gleich $(n-1)*(g_{re}-1)$, kurz:

$$|V_{Z_\alpha}(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])} F_{nw(\alpha)}(\alpha[i]))| \leq (n-1)*(g_{re}-1).$$

Beweis: Angenommen der Prozeß P_i von P muß im Zustand $val_P(\alpha)$ zum Wiederaufsetzpunkt $WP_\alpha(\alpha[k])$ zurückgesetzt werden. Dann sei $\alpha[p_j]$ die älteste Aktion des Prozesses P_j, deren Zurücksetzung unvermeidbar ist. Nun betrachte man eine Aktion $\alpha[l]$, die zum Prozeß P_j gehört und im Intervall $[t(WP_\alpha(\alpha[p_j])), t(\alpha[p_j])[$ ausgeführt wurde. Für $\alpha[l]$ gilt insbesondere:

$$WP_\alpha(\alpha[l]) = WP_\alpha(\alpha[p_j])$$

Eine solche Aktion $\alpha[l]$ kann es (muß es nicht!) in jedem beliebigen Prozeß P_j von P, $j \neq i$, geben.
Für ein derartiges $\alpha[l]$ gilt selbstverständlich $\alpha[l](B_\alpha \cup E_\alpha)\alpha[p_j]B_\alpha^*\alpha[p_i]$, so daß $\alpha[l]$ in Folge der Zurücksetzung von P_i zum Wiederaufsetzpunkt $WP_\alpha(\alpha[k])$ zurückgesetzt werden muß. Da jedoch nach Annahme $proc_P(\alpha[l], \alpha[p_j]) \;\&\; t(\alpha[l]) < t(\alpha[p_j])$ gilt, ist die Zurücksetzung von $\alpha[l]$ nicht unvermeidbar.
Nach Voraussetzung ist des weiteren:

$$|\{\alpha[l] \mid t(\alpha[l]) < t(\alpha[p_j]) \;\&\; proc_P(\alpha[l], \alpha[p_j]) \;\&\; WP_\alpha(\alpha[l]) = WP_\alpha(\alpha[p_j])\}| < g_{re},$$

und daher kann es im Prozeß P_j, $1 \leq j \leq n \;\&\; i \neq j$, maximal $g_{re} - 1$ solcher Aktionen $\alpha[l]$ geben. Um zu zeigen, daß es neben den maximal $g_{re} - 1$ Aktionen eines Prozesses P_j, $1 \leq j \leq n$, die im Intervall $[t(WP_\alpha(\alpha[p_j])), t(\alpha[p_j])]$ zur Ausführung kamen, keine weiteren Aktionen gibt, deren Zurücksetzung vermeidbar wäre, sei angenommen, daß es eine Aktion $\alpha[m] \in W(\alpha)$ gibt, $proc_P(\alpha[p_j], \alpha[m])$, die vor dem Zeitpunkt $t(WP_\alpha(\alpha[p_j]))$ ausgeführt wurde, und die zurückgesetzt werden muß. In diesem Falle muß $\alpha[m](B_\alpha \cup E_\alpha)^*\alpha[p_i]$ gelten und nach Hilfssatz 3.2.4/5 muß eine endliche Folge $\alpha[l_1], \ldots, \alpha[l_m]$ existieren (aufgrund der Annahme:

$$\exists \alpha[m] \in W(\alpha): [proc_P(\alpha[p_j], \alpha[m]) \;\&\; t(\alpha[m]) < t(WP_\alpha(\alpha[p_j]))]$$

muß $m > 0$ sein), so daß gilt:

$$\alpha[m] = \alpha[l_m](B_\alpha \cup E_\alpha)\alpha[l_{m-1}] \ldots \alpha[l_1](B_\alpha \cup E_\alpha)\alpha[p_i].$$

Wie im Satz 5.1.2/1 begründet, muß gelten:

$$(**)\; \alpha[l_1] = \alpha[p_k] \;\|\; \exists k \in \{1, \ldots, n\}: [t(\alpha[p_k]) < t(\alpha[l_1]) \;\&\; proc_P(\alpha[p_k], \alpha[l_1])].$$

Betrachtet man nun die Beziehung $\alpha[l_l](B_\alpha \cup E_\alpha)\alpha[l_{l-1}]$, $1 < l \leq m$, unter der Annahme $t(\alpha[l_l]) < t(WP_\alpha(\alpha[p_j])) \;\&\; proc_P(\alpha[p_j], \alpha[l_l])$, so sind zwei Fälle zu unterscheiden.

Fall 1 $(\alpha[l_l]E_\alpha\alpha[l_{l-1}])$: Aufgrund der gemachten Annahme

$$t(\alpha[l_l]) < t(WP_\alpha(\alpha[p_j])) \;\&\; proc_P(\alpha[p_j], \alpha[l_l])$$

und, da $\alpha[l_l]E_\alpha\alpha[l_{l-1}]$ gilt, folgt in diesem Fall

$$t(\alpha[l_{l-1}]) < t(WP_\alpha(\alpha[p_j])) \;\&\; proc_P(\alpha[l_{l-1}], \alpha[p_j]).$$

Fall 2 $(\alpha[l_l]B_\alpha\alpha[l_{l-1}])$: In diesem Fall sind wiederum zwei Fälle zu unterscheiden:

Fall 2.1 ($proc_P(\alpha[l_i],\alpha[l_{i-1}])$): Nach Definition von $\alpha[l_i]B_a\alpha[l_{i-1}]$ gilt $t(\alpha[l_{i-1}])<t(\alpha[l_i])$. Aufgrund der gemachten Annahme $t(\alpha[l_i])<t(WP_a(\alpha[p_j]))$ & $proc_P(\alpha[p_j],\alpha[l_i])$ und, da $proc_P(\alpha[l_i],\alpha[l_{i-1}])$ gilt, folgt somit

$$t(\alpha[l_{i-1}])<t(\alpha[l_i])<t(WP_a(\alpha[p_j])) \;\&\; proc_P(\alpha[l_{i-1}],\alpha[p_j]).$$

Fall 2.2 ($\neg proc_P(\alpha[l_i],\alpha[l_{i-1}])$): Da $\alpha[l_i]$ und $\alpha[l_{i-1}]$ nicht zum gleichen Prozeß gehören, gilt aufgrund der Definition von $\alpha[l_i]B_a\alpha[l_{i-1}]$: $gS(\alpha[l_{i-1}])$=**true**, d.h. $\alpha[l_{i-1}]$ ist eine globale Schreibaktion
Angenommen $\alpha[l_{i-1}]$ gehört zum Prozeß P_q und es gilt $t(WP_a(\alpha[p_q]))\leq t(\alpha[l_{i-1}])$, $1\leq q\leq n$. Nach Hilfssatz 5.1.1/1 muß dann entweder $q=i$ oder $gL(\alpha[p_q])$=**true** sein.
Da im Falle $q=i$ die Zurücksetzung von $\alpha[l_{i-1}]$ unvermeidbar ist und wegen der Beziehung $\alpha[l_i]B_a\alpha[l_{i-1}]$ damit auch die Zurücksetzung von $\alpha[l_i]$ nicht vermieden werden kann im Widerspruch zur Annahme $t(\alpha[l_i])<t(WP_a(\alpha[p_j]))$ & $proc_P(\alpha[p_j],\alpha[l_i])$, muß $i\neq q$ sein
Zu prüfen bleibt, ob $gL(\alpha[p_q])$ gelten kann. $gL(\alpha[p_q])$=**true** bedeutet wegen (*), daß $t(\alpha[p_q])<t(\alpha[l_{i-1}])$ (*diese Folgerung wird am Ende des Beweises noch näher begründet*), und wegen der Annahme $proc_P(\alpha[p_q],\alpha[l_{i-1}])$ heißt das aber, daß die Zurücksetzung von $\alpha[l_{i-1}]$ auch in diesem Falle unvermeidbar ist, und daß die Zurücksetzung von $\alpha[l_i]$, da $\alpha[l_i]B_a\alpha[l_{i-1}]$ gilt, auch nicht vermieden werden kann.
Unter der Annahme $t(\alpha[l_i])<t(WP_a(\alpha[p_j]))$ & $proc_P(\alpha[l_i],\alpha[p_j])$ kann im Fall 2.2 nur $t(\alpha[l_{i-1}])<t(WP_a(\alpha[p_q]))$ & $proc_P(\alpha[l_{i-1}],\alpha[p_q])$ für ein q, $1\leq q\leq n$, gelten.

In den zu unterscheidenden Fällen 1 und 2 gilt also

$$\exists j\in\{1,\ldots,n\}: [t(\alpha[l_{i-1}])<t(WP_a(\alpha[p_j])) \;\&\; proc_P(\alpha[l_{i-1}],\alpha[p_j])].$$

Mit Hilfe vollständiger Induktion folgt, daß für ein k, $1\leq k\leq n$,

$$t(\alpha[l_1])\leq t(WP_a(\alpha[p_k])) \;\&\; proc_P(\alpha[l_1],\alpha[p_k])$$

gelten muß. Da per Definition $t(WP_a(\alpha[p_k]))<t(\alpha[p_k])$, stellt dies einen Widerspruch zu (**) dar. Insgesamt folgt, daß $t(\alpha[m])<t(WP_a(\alpha[p_k]))$ & $proc_P(\alpha[m],\alpha[p_k])$, $1\leq k\leq n$, nicht gelten kann.

Damit ist gezeigt, daß in jedem Prozeß P_j, $1\leq j\leq n$, $j\neq i$, maximal $g_{ro}-1$ Aktionen bei Zurücksetzung von Prozeß P_i zum Wiederaufsetzpunkt $WP_a(\alpha[k])=WP_a(\alpha[p_i])$ rückgängig gemacht werden müssen, deren Zurücksetzung vermieden werden könnte. Da die Zurücksetzung aller Aktionen des Prozesses P_i unvermeidbar ist (im Intervall $[t(WP_a(\alpha[k])),t(\alpha[p_i])]$ werden keine Aktionen des Prozesses P_i ausgeführt), läßt sich die Zahl der Aktionen, die im Falle der Zurücksetzung des Prozesses P_i zum Wiederaufsetzpunkt $WP_a(\alpha[k])$ im Zusatnd $val_P(\alpha)$ zurückgesetzt werden müssen und deren Zurücksetzung vermeidbar wäre, auf maximal $(n-1)*(g_{ro}-1)$ beschränken.

Erläuterung: Von

$$gL(\alpha[p_q]) \;\&\; gS(\alpha[l_{i-1}]) \;\&\; proc_P(\alpha[p_q],\alpha[l_{i-1}]) \;\&\; t(WP_a(\alpha[p_q]))\leq t(\alpha[l_{i-1}])$$

und der Voraussetzung (*) kann aus folgendem Grunde auf $t(\alpha[p_q])\leq t(\alpha[l_{i-1}])$

geschlossen werden. Wäre $t(\alpha[l_{i-1}])<t(\alpha[p_q])$, so würde, da $proc_P(\alpha[p_q],\alpha[l_{i-1}])$ und $gS(\alpha[l_{i-1}])$ sowie $gL(\alpha[p_q])$ gelten, auf $WP_\alpha(\alpha[l_{i-1}])\neq WP_\alpha(\alpha[p_q])$ geschlossen werden können. Da jedoch in diesem Falle auch

$$t(WP_\alpha(\alpha[p_q]))\leq t(\alpha[l_{i-1}])<t(\alpha[p_q])$$

gelten würde, könnte wegen $proc_P(\alpha[p_q],\alpha[l_{i-1}])$ auf $WP_\alpha(\alpha[l_{i-1}])=WP_\alpha(\alpha[p_q])$ geschlossen werden, im Widerspruch zu der vorangehenden Folgerung. ■

Satz 5.1.2/3: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1,\ldots,P_n$, in dem für jeden aktiven Prozeßablauf α in bezug auf jeden Prozeß P_i, $1\leq i\leq n$, gilt:

$$(*)\ \forall\alpha[i],\alpha[j],\alpha[k]\subset W(\alpha).\ [(\ proc_P(\alpha[i],\alpha[j])\ \&\ \alpha[j]B_\alpha\alpha[i]\ \&\ proc_P(\alpha[j],\alpha[k])\ \&\ t(\alpha[k])<t(\alpha[j])\ \&\ gS(\alpha[k])\ \&\ gL(\alpha[j]))\ \Rightarrow\ (WP_\alpha(\alpha[j])\neq WP_\alpha(\alpha[k]))]$$

Des weiteren existiere eine *maximale Recoveryeinheitsgröße* $g_{re}\in\mathbf{N}$, so daß für alle aktiven Berechnungen α von P gilt:

$$g_{re_\alpha}::=\max_{\alpha[k]\in W(\alpha)}|\{\alpha[j]\mid\alpha[j]\in W(\alpha)\ \&\ \alpha[j]E_\alpha\alpha[k]\}|<g_{re}.$$

Dann ist die Anzahl der beim Zurücksetzen eines Prozesses P_i von P zum Wiederaufsetzpunkt $WP_\alpha(\alpha[k])$ im Zustand $val_P(\alpha)$ zurückzusetzenden Aktionen, deren Zurücksetzung vermeidbar wäre, kleiner oder gleich $(n-1)*(g_{re}-1)$, kurz:

$$|V_{Z_\alpha}(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{re_W(\alpha)}(\alpha[i]))|\leq(n-1)*(g_{re}-1).$$

Beweis: Angenommen der Prozeß P_i von P muß im Zustand $val_P(\alpha)$ zum Wiederaufsetzpunkt $WP_\alpha(\alpha[k])$ zurückgesetzt werden. Dann sei $\alpha[p_j]$ die älteste Aktion des Prozesses P_j, deren Zurücksetzung unvermeidbar ist. Nun betrachte man eine Aktion $\alpha[l]$, die zum Prozeß P_j gehört und im Intervall $[t(WP_\alpha(\alpha[p_j])),t(\alpha[p_j])[$ ausgeführt wurde. Für $\alpha[l]$ gilt insbesondere:

$$WP_\alpha(\alpha[l])=WP_\alpha(\alpha[p_j]).$$

Eine solche Aktion $\alpha[l]$ kann es (muß es nicht!) in jedem beliebigen Prozeß P_j von P, $j\neq i$, geben.
Für ein derartiges $\alpha[l]$ gilt selbstverständlich $\alpha[l](B_\alpha\cup E_\alpha)\alpha[p_j]B_\alpha^*\alpha[p_i]$, so daß $\alpha[l]$ in Folge der Zurücksetzung von P_i zum Wiederaufsetzpunkt $WP_\alpha(\alpha[k])$ zurückgesetzt werden muß. Da jedoch nach Annahme $proc_P(\alpha[l],\alpha[p_j])\ \&\ t(\alpha[l])<t(\alpha[p_j])$ gilt, ist die Zurücksetzung von $\alpha[l]$ nicht unvermeidbar.
Nach Voraussetzung ist des weiteren:

$$|\{\alpha[l]\mid t(\alpha[l])<t(\alpha[p_j])\ \&\ proc_P(\alpha[l],\alpha[p_j])\ \&\ WP_\alpha(\alpha[l])=WP_\alpha(\alpha[p_j])\}|<g_{re}$$

und daher kann es im Prozeß P_j, $1\leq j\leq n\ \&\ i\neq j$, maximal $g_{re}-1$ solcher Aktionen $\alpha[l]$ geben. Um zu zeigen, daß es neben den maximal $g_{re}-1$ Aktionen eines Prozesses P_j,

$1 \le j \le n$, die im Intervall $[t(WP_a(\alpha[p_j])), t(\alpha[p_j])]$ zur Ausführung kamen, keine weiteren Aktionen gibt, deren Zurücksetzung vermeidbar wäre, sei angenommen, daß es eine Aktion $\alpha[m] \in W(\alpha)$ gibt, $proc_P(\alpha[p_j], \alpha[m])$, die vor dem Zeitpunkt $t(WP_a(\alpha[p_j]))$ ausgeführt wurde, und die zurückgesetzt werden muß. In diesem Falle muß $\alpha[m](B_a \cup E_a)^* \alpha[p_i]$ gelten und nach Hilfssatz 3.2.4/5 muß eine endliche Folge $\alpha[l_1], \ldots, \alpha[l_m]$ existieren (aufgrund der Annahme

$$\exists \alpha[m] \in W(\alpha): [proc_P(\alpha[p_j], \alpha[m]) \ \& \ t(\alpha[m]) < t(WP_a(\alpha[p_j]))]$$

muß $m > 0$ sein), so daß gilt:

$$\alpha[m] = \alpha[l_m](B_a \cup E_a)\alpha[l_{m-1}] \ldots \alpha[l_1](B_a \cup E_a)\alpha[p_i].$$

Wie im Satz 5.1.2/1 begründet, muß gelten

$$(**)\ \alpha[l_1] = \alpha[p_k] \ \| \ \exists k \in \{1, \ldots, n\}: [t(\alpha[p_k]) < t(\alpha[l_1]) \ \& \ proc_P(\alpha[p_k], \alpha[l_1])].$$

Betrachtet man nun die Beziehung $\alpha[l_l](B_a \cup E_a)\alpha[l_{l-1}]$, $1 < l \le m$, unter der Annahme $t(\alpha[l_l]) < t(WP_a(\alpha[p_j])) \ \& \ proc_P(\alpha[p_j], \alpha[l_l])$, so sind zwei Fälle zu unterscheiden:

Fall 1 $(\alpha[l_l] E_a \alpha[l_{l-1}])$: Aufgrund der gemachten Annahme

$$t(\alpha[l_l]) < t(WP_a(\alpha[p_j])) \ \& \ proc_P(\alpha[p_j], \alpha[l_l])$$

und, da $\alpha[l_l] E_a \alpha[l_{l-1}]$ gilt, folgt in diesem Fall

$$t(\alpha[l_{l-1}]) < t(WP_a(\alpha[p_j])) \ \& \ proc_P(\alpha[l_{l-1}], \alpha[p_j]).$$

Fall 2 $(\alpha[l_l] B_a \alpha[l_{l-1}])$: In diesem Fall sind wiederum zwei Fälle zu unterscheiden.

Fall 2.1 $(proc_P(\alpha[l_l], \alpha[l_{l-1}]))$: Nach Definition von $\alpha[l_l] B_a \alpha[l_{l-1}]$ gilt $t(\alpha[l_{l-1}]) < t(\alpha[l_l])$. Aufgrund der gemachten Annahme $t(\alpha[l_l]) < t(WP_a(\alpha[p_j])) \ \& \ proc_P(\alpha[p_j], \alpha[l_l])$ und, da $proc_P(\alpha[l_l], \alpha[l_{l-1}])$ gilt, folgt somit:

$$t(\alpha[l_{l-1}]) < t(\alpha[l_l]) < t(WP_a(\alpha[p_j])) \ \& \ proc_P(\alpha[l_{l-1}], \alpha[p_j]).$$

Fall 2.2 $(\neg proc_P(\alpha[l_l], \alpha[l_{l-1}]))$: Da $\alpha[l_l]$ und $\alpha[l_{l-1}]$ nicht zum gleichen Prozeß gehören, gilt aufgrund der Definition von $\alpha[l_l] B_a \alpha[l_{l-1}]$: $gS(\alpha[l_{l-1}]) = \textbf{true}$, d.h. $\alpha[l_{l-1}]$ ist eine globale Schreibaktion.
Angenommen $\alpha[l_{l-1}]$ gehört zum Prozeß P_q und es gilt $t(WP_a(\alpha[p_q])) \le t(\alpha[l_{l-1}])$, $1 \le q \le n$. Nach Hilfssatz 5.1.1/1 muß dann entweder $q = i$ oder $gL(\alpha[p_q]) = \textbf{true}$ sein.
Da im Falle $q = i$ die Zurücksetzung von $\alpha[l_{l-1}]$ unvermeidbar ist und wegen der Beziehung $\alpha[l_l] B_a \alpha[l_{l-1}]$ damit auch die Zurücksetzung von $\alpha[l_l]$ nicht vermieden werden kann im Widerspruch zur Annahme $t(\alpha[l_l]) < t(WP_a(\alpha[p_j])) \ \& \ proc_P(\alpha[p_j], \alpha[l_l])$, muß $i \neq q$ sein.
Zu prüfen bleibt, ob $gL(\alpha[p_q])$ gelten kann: $gL(\alpha[p_q]) = \textbf{true}$ bedeutet wegen (*), daß $t(\alpha[p_q]) \le t(\alpha[l_{l-1}])$ (*diese Folgerung wird am Ende des Beweises noch näher begründet*), und wegen der Annahme $proc_P(\alpha[p_q], \alpha[l_{l-1}])$ heißt das aber, daß die Zurücksetzung von $\alpha[l_{l-1}]$ auch in diesem Falle unvermeidbar ist, und daß die

Zurücksetzung von $\alpha[l_i]$, da $\alpha[l_i]B_a\alpha[l_{i-1}]$ gilt, auch nicht vermieden werden kann. Unter der Annahme $t(\alpha[l_i])<t(WP_a(\alpha[p_j]))$ & $proc_P(\alpha[l_i],\alpha[p_j])$ kann im Fall 2.2 nur $t(\alpha[l_{i-1}])<t(WP_a(\alpha[p_q]))$ & $proc_P(\alpha[l_{i-1}],\alpha[p_q])$ für ein q, $1\leq q\leq n$, gelten.

In den zu unterscheidenden Fällen 1 und 2 gilt also:

$$\exists j\in\{1,\ldots,n\}\colon [t(\alpha[l_{i-1}])<t(WP_a(\alpha[p_j]))\ \&\ proc_P(\alpha[l_{i-1}],\alpha[p_j])].$$

Mit Hilfe vollständiger Induktion folgt, daß für ein k, $1\leq k\leq n$,

$$t(\alpha[l_1])\leq t(WP_a(\alpha[p_k]))\ \&\ proc_P(\alpha[l_1],\alpha[p_k])$$

gelten muß. Da per Definition $t(WP_a(\alpha[p_k]))<t(\alpha[p_k])$, stellt dies einen Widerspruch zu (**) dar. Insgesamt folgt, daß $t(\alpha[m])<t(WP_a(\alpha[p_k]))$ & $proc_P(\alpha[m],\alpha[p_k])$, $1\leq k\leq n$, nicht gelten kann.

Damit ist gezeigt, daß in jedem Prozeß P_j, $1\leq j\leq n$, $j\neq i$, maximal $g_{\tau_0}-1$ Aktionen bei Zurücksetzung von Prozeß P_i zum Wiederaufsetzpunkt $WP_a(\alpha[k])=WP_a(\alpha[p_i])$ rückgängig gemacht werden müssen, deren Zurücksetzung vermieden werden könnte. Da die Zurücksetzung aller Aktionen des Prozesses P_i unvermeidbar ist (im Intervall $[t(WP_a(\alpha[k])),t(\alpha[p_i])]$ werden keine Aktionen des Prozesses P_i ausgeführt), läßt sich die Zahl der Aktionen, die im Falle der Zurücksetzung des Prozesses P_i zum Wiederaufsetzpunkt $WP_a(\alpha[k])$ im Zusatnd $val_P(\alpha)$ zurückgesetzt werden müssen und deren Zurücksetzung vermeidbar wäre, auf maximal $(n-1)*(g_{\tau_0}-1)$ beschränken.

Erläuterung: Von

$$gL(\alpha[p_q])\ \&\ gS(\alpha[l_{i-1}])\ \&\ proc_P(\alpha[p_q],\alpha[l_{i-1}])\ \&\ t(WP_a(\alpha[p_q]))\leq t(\alpha[l_{i-1}])$$

und der Voraussetzung (*) des Satzes kann aus folgendem Grund auf $t(\alpha[p_q])\leq t(\alpha[l_{i-1}])$ geschlossen werden: In dem betrachteten Falle $i\neq q$ muß nach Hilfssatz 5.1.1/1 eine Aktion $\alpha[r]\in W(\alpha)$ existieren, so daß gilt:

$$\alpha[p_q]B_a\alpha[r]\ \&\ proc_P(\alpha[r],\alpha[p_q]).$$

Da nach Annahme

$$proc_P(\alpha[p_q],\alpha[l_{i-1}])\ \&\ gS(\alpha[l_{i-1}])\ \&\ gL(\alpha[p_q])$$

gilt, kann $t(\alpha[l_{i-1}])<t(\alpha[p_q])$ nicht erfüllt sein, denn wegen der Voraussetzung (*) würde in diesem Falle $WP_a(\alpha[p_q])\neq WP_a(\alpha[l_{i-1}])$ und des weiteren, wegen $t(\alpha[l_{i-1}])<t(\alpha[p_q])$, auch $t(WP_a(\alpha[l_{i-1}]))<t(WP_a(\alpha[p_q]))$ gelten, im Widerspruch zur Annahme $t(WP_a(\alpha[p_q]))\leq t(WP_a(\alpha[l_{i-1}]))$. ■

Satz 5.1.2/4: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen. Jede aktive Berechnung α von P besitze entweder die in Satz 5.1.2/1, Satz 5.1.2/2 oder Satz 5.1.2/3 beschriebenen Eigenschaften. Des weiteren gebe es Konstanten $n_1,n_2\in\mathbb{N}$, so daß für jede aktive Berechnung α von P gelte:

(1) $\forall \alpha[j] \subset W(\alpha): \{i \mid \alpha[i] \subset W(\alpha) \ \& \ \alpha[j] E_a \alpha[i]\} < n_1$ und

(2) in bezug auf alle als fehlerhaft erkannten Aktionen $\alpha[j] \in W(\alpha)$:

$$|Z_a(\bigcup_{\alpha[i] \in W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(WP_a(\alpha[j])), t(\alpha[|\alpha|])])| < n_2.$$

Unter diesen Voraussetzungen läßt sich zeigen

$$|Z_a(\bigcup_{\alpha[i] \in W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))| < n^* (n_1 - 1) + n_2.$$

Beweis: Die Größe von $|Z_a(\bigcup_{\alpha[i] \subset W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))|$ läßt sich wie folgt abschätzen (im folgenden wird t_1, t_2, $t_1 < t_2$, abkürzend für das Zeitintervall $[t_1, t_2]$ ohne den Zeitpunkt t_2 geschrieben)

$$|Z_a(\bigcup_{\alpha[i] \in W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))| \leq |Z_a(\bigcup_{\alpha[i] \in W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[1]), t(WP_a(\alpha[j]))[)| +$$

$$|Z_a(\bigcup_{\alpha[i] \in W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(WP_a(\alpha[j])), t(\alpha[|\alpha|])])| <$$

$$|Z_a(\bigcup_{\alpha[i] \in W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[1]), t(WP_a(\alpha[j]))[)| + n_2,$$

für eine als fehlerhaft erkannte Aktion $\alpha[j] \in W(\alpha)$, $t(\alpha[j])$ minimal.

Bleibt zu zeigen

$$|Z_a(\bigcup_{\alpha[i] \in W(\alpha) \& f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[1]), t(WP_a(\alpha[j]))[)| \leq n^* (n_1 - 1).$$

Letzteres folgt unmittelbar mit den Aussagen der Sätze 5.1.2/1, 2 bzw. 3. ■

5.2. Wiederaufsetzpunktgesicherte Prozeßwechsel

Wie im folgenden gezeigt wird, laßt sich das Problem der unbeschränkbaren zyklischen Zurücksetzung nicht nur mittels der Technik der Sicherung globaler Leseaufrufe durch Wiederaufsetzpunkte, sondern auch durch prozeßwechselbezogene Sicherungsprinzipien in den Griff bekommen. In diesem Fall wird vor einem Prozeßwechsel entweder im verdrängten oder im verdrängenden Prozeß ein Wiederaufsetzpunkt eingerichtet. Durch welche Merkmale sich prozeßwechselbezogen gesicherte Aktionen charakterisieren lassen, zeigt der folgende Abschnitt.

5.2.1. Charakterisierung prozeßwechselbezogener Sicherungen

Prozeßwechselbezogene Sicherungen, die im Falle einer Prozeßverdrängung, Prozessorfreigabe oder Prozeßfortsetzung eingerichtet werden, lassen sich durch folgende Bedingungen treffend beschreiben.

Hilfssatz 5.2.1/1: Sei $P=(A,D,w)$ ein (auf einem Monoprozessor ablaufendes) Prozeßsystem und der Ausführungsteil von Aktionen $a \in A$ sei nicht unterbrechbar. Dann gilt für beliebige Aktionen $\alpha[i],\alpha[j],\alpha[k] \in W(\alpha)$:

(1) Wird bei *Verdrängung eines Prozesses* ein Wiederaufsetzpunkt bzgl. dieses Prozesses gesetzt, so kann von

$$\alpha[k]E_a^*\alpha[j]B_a\alpha[i] \;\&\; \neg proc_P(\alpha[i],\alpha[j]) \text{ auf } t(\alpha[k])>t(\alpha[i])$$

geschlossen werden.

(2) Wird bei *Fortsetzung eines Prozesses* ein Wiederaufsetzpunkt bzgl. dieses Prozesses gesetzt, so kann von

$$\alpha[j]B_a\alpha[i] \;\&\; \neg proc_P(\alpha[i],\alpha[j]) \text{ auf } t(WP_a(\alpha[j]))>t(\alpha[i])$$

geschlossen werden.

(3) Wird ohne Verdrängung gearbeitet und ein Wiederaufsetzpunkt bzgl. eines Prozesses immer dann gesetzt, wenn dieser Prozeß den Prozessor freigibt, so gilt ebenfalls die Aussage (1), d.h.:

$$(\alpha[k]E_a^*\alpha[j]B_a\alpha[i] \;\&\; \neg proc_P(\alpha[i],\alpha[j])) \Rightarrow t(\alpha[k])>t(\alpha[i]).$$

Beweis:

zu(1): Angenommen nein. Dann muß aufgrund der Voraussetzungen $\alpha[k]E_a^*\alpha[j]B_a\alpha[i]$

und $\neg proc_P(\alpha[i],\alpha[j])$ gelten: $t(\alpha[k])<t(\alpha[i])<t(\alpha[j])$. Das bedeutet aber: Prozeß $P_{bz(\alpha[k])}$ wurde durch den Prozeß $P_{bz(\alpha[i])}$ verdrängt, ohne daß ein Wiederaufsetzpunkt bzgl. des Prozesses $P_{bz(\alpha[k])}$ gesetzt wurde ($\alpha[j]$ und $\alpha[k]$ gehören ja nach Voraussetzung zur gleichen Recoveryeinheit). Dies steht im Widerspruch zur Voraussetzung (1), daß bei Verdrängung eines Prozesses ein Wiederaufsetzpunkt bzgl. dieses Prozesses gesetzt wird.

zu(2) Angenommen nein. Dann muß aufgrund der Voraussetzungen $\alpha[j]B_a\alpha[i]$ und $\neg proc_P(\alpha[i],\alpha[j])$ gelten: $t(\alpha[j])>t(\alpha[i])>t(WP_a(\alpha[j]))$. Das bedeutet aber, daß Prozeß $P_{bz(\alpha[j])}$ unterbrochen wird im Intervall $[t(WP_a(\alpha[j])),t(\alpha[j])]$ und fortgesetzt wird, ohne daß ein Wiederaufsetzpunkt bzgl. des Prozesses $P_{bz(\alpha[j])}$ bei Fortsetzung von $P_{bz(\alpha[j])}$ gesetzt wurde. Dies steht im Widerspruch zur Voraussetzung von (2).

zu(3) Angenommen nein. Dann muß aufgrund der Voraussetzungen $\alpha[k]E_a^*\alpha[j]B_a\alpha[i]$ und $proc_P(\alpha[i],\alpha[j])$ gelten $t(\alpha[k])<t(\alpha[i])<t(\alpha[j])$. Das bedeutet aber: Prozeß $P_{bz(\alpha[k])}$ hat den Prozessor freigegeben, ohne daß ein Wiederaufsetzpunkt bzgl. des Prozesses $P_{bz(\alpha[k])}$ gesetzt wurde ($\alpha[j]$,$\alpha[k]$ gehören zur gleichen Recoveryeinheit!). Dies steht im Widerspruch zur Voraussetzung (3), daß bei Freigabe eines Prozessors ein Wiederaufsetzpunkt bzgl. des den Prozessor freigebenden Prozesses gesetzt wird. ∎

5.2.2. Wirksamkeit prozeßwechselbezogener Sicherungen

Basierend auf dem Hilfssatz 5.2.1/1 werden die in diesem Abschnitt abgeleiteten Aussagen zeigen, daß, falls bei der Verdrängung eines Prozesses durch einen anderen Prozeß oder bei der Fortsetzung eines Prozesses nach einem Prozeßwechsel jeweils ein Wiederaufsetzpunkt im verdrängten, freigebenden bzw. fortgesetzten Prozeß eingerichtet wird, alle im Fehlerfall einer Aktion $\alpha[i]$ zurückzusetzenden Aktionen nicht vor dem Zeitpunkt $t(WP_a(\alpha[i]))$ und alle zurückzusetzenden Aktionen fremder Prozesse sogar nicht vor dem Zeitpunkt $t(\alpha[i])$ ausgeführt worden sein konnten (vgl. Satz 5.2.2/1 bzw. Satz 5.2.2/2). Des weiteren wird auf diese Weise gezeigt, daß durch prozeßwechselbezogene Sicherungen die zyklische Zurücksetzung und damit, im Falle berechnungsunabhängig beschränkbarer Recoveryeinheiten und Prozeßinaktivphasen, auch der Dominoeffekt prinzipiell vermieden/verhindert werden kann (vgl. Satz 5.2.2/3).

Satz 5.2.2/1: Sei $P=(A,D,w)$ ein (auf einem Monoprozessor ablaufendes) Prozeßsystem und α eine aktive Berechnung in P. Für alle Aktionen $\alpha[i],\alpha[j],\alpha[k]\in W(\alpha)$ gelte:

$$(*)\ (\alpha[k]E_a^*\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow\ t(\alpha[k])>t(\alpha[i]).$$

In diesem Falle läßt sich zeigen, daß die Zurücksetzung einer Aktion $\alpha[i]\in W(\alpha)$ nur die Zurücksetzung von Aktionen $\alpha[j]\in W(\alpha)$, die nach dem Zeitpunkt $t(WP_a(\alpha[i]))$ ausgeführt wurden ($t(\alpha[j])\geq t(WP_a(\alpha[i]))$), erforderlich machen kann.

Beweis: Für jede Aktion $\alpha[j]$, die bei Zurücksetzung von $\alpha[i]$ zurückgesetzt werden muß, gilt $\alpha[j](E_\alpha \cup B_\alpha)^* \alpha[i]$. Daher muß es nach Hilfssatz 3.2.4/5 Aktionen $\alpha[j_1],\ldots,\alpha[j_m] \in W(\alpha)$ geben mit der Eigenschaft:

$$\alpha[j]=\alpha[j_m](E_\alpha \cup B_\alpha)\alpha[j_{m-1}]\ldots\alpha[j_2](E_\alpha \cup B_\alpha)\alpha[j_1]=\alpha[i].$$

Zum Beweis des Satzes genügt es nun zu zeigen, daß es keine Aktion $\alpha[j_l]$ geben kann, für die gilt: $t(\alpha[j_l])<t(WP_\alpha(\alpha[i]))$.
Angenommen es gäbe so eine Aktion $\alpha[j_l]$; sei l minimal gewählt. Sei $\alpha[j_{l-k}]$ eine Aktion, für die gilt: $\alpha[j_l]E_\alpha^* \alpha[j_{l-k+1}]B_\alpha \alpha[j_{l-k}]$ & $\neg proc_P(\alpha[j_l],\alpha[j_{l-k}])$, wobei $l>k\geq 1$. Je nach dem, ob ein $\alpha[j_{l-k}]$ mit dieser Eigenschaft existiert oder nicht, sind zwei Fälle zu unterscheiden:

Fall 1 ($\alpha[j_{l-k}]$ existiert) Unter der Voraussetzung (*) folgt in diesem Fall (vgl. Hilfssatz 5.2.1/1) $t(\alpha[j_l])>t(\alpha[j_{l-k}])$ im Widerspruch zur Minimalität von l, d.h. Fall 1 kann nicht auftreten.

Fall 2 ($\alpha[j_{l-k}]$ existiert nicht) In diesem Falle muß (falls $l=1$ ist trivialerweise, falls $l>1$ ist aufgrund der folgenden Beziehungen) $\alpha[j_l]E_\alpha\alpha[i]$ gelten, denn von

$$\alpha[j_l]E_\alpha^*\alpha[j_{l-k+1}]B_\alpha\alpha[j_{l-k}] \;\&\; proc_P(\alpha[j_l],\alpha[j_{l-k}])$$

kann aufgrund der Beziehung $t(\alpha[j_l])<t(\alpha[j_{l-k}])<t(\alpha[j_{l-k+1}])$ und der Beziehung $\alpha[j_l]E_\alpha\alpha[j_{l-k+1}]$ auf $\alpha[j_l]E_\alpha\alpha[j_{l-k}]$ geschlossen werden ($t(\alpha[j_l])<t(\alpha[j_{l-k}])$ kann aufgrund der Wahl von l gefolgert werden). $\alpha[j_l]E_\alpha\alpha[i]$ folgt schließlich mit vollständiger Induktion.
Mit $\alpha[j_l]E_\alpha\alpha[i]$ folgt jedoch $t(\alpha[j_l])\geq t(WP_\alpha(\alpha[i]))$, was einen Widerspruch zur Annahme $t(\alpha[j_l])<t(WP_\alpha(\alpha[i]))$ darstellt. ■

Satz 5.2.2/2: Sei $P=(A,D,w)$ ein (auf einem Monoprozessor ablaufendes) Prozeßsystem und α eine aktive Berechnung von P. Für alle Aktionen $\alpha[i],\alpha[j] \in W(\alpha)$ gelte:

$$(*)\ (\alpha[j]B_\alpha\alpha[i] \;\&\; \neg proc_P(\alpha[i],\alpha[j])) \;\Rightarrow\; t(WP_\alpha(\alpha[j]))>t(\alpha[i]).$$

In diesem Falle läßt sich zeigen, daß die Zurücksetzung einer Aktion $\alpha[i] \in W(\alpha)$ nur die Zurücksetzung von Aktionen $\alpha[j] \in W(\alpha)$, die nach dem Zeitpunkt $t(WP_\alpha(\alpha[i]))$ ausgeführt wurden ($t(\alpha[j])\geq t(WP_\alpha(\alpha[i]))$), erforderlich machen kann.

Beweis: Muß die Aktion $\alpha[j]$ bei Zurücksetzung einer Aktion $\alpha[i]$ ebenfalls zurückgesetzt werden, so muß $\alpha[j](E_\alpha \cup B_\alpha)^*\alpha[i]$ gelten. Daher müssen nach Hilfssatz 3.2.4/5 Aktionen $\alpha[j_1],\ldots,\alpha[j_m] \in W(\alpha)$ existieren mit der Eigenschaft:

$$\alpha[j]=\alpha[j_m](E_\alpha \cup B_\alpha)\alpha[j_{m-1}]\ldots\alpha[j_2](E_\alpha \cup B_\alpha)\alpha[j_1]=\alpha[i].$$

Zum Beweis des Satzes genügt es nun zu zeigen, daß es keine Aktion $\alpha[j_l]$ geben kann, für die $t(\alpha[j_l])<t(WP_\alpha(\alpha[i]))$ gilt.
Angenommen es gäbe eine solche Aktion $\alpha[j_l]$; sei l minimal gewählt. Sei $\alpha[j_{l-k}]$ eine Aktion für die gilt $\alpha[j_l]E_\alpha^*\alpha[j_{l-k+1}]B_\alpha\alpha[j_{l-k}]$ & $\neg proc_P(\alpha[j_l],\alpha[j_{l-k}])$, wobei $l>k\geq 1$.

Zwei Fälle sind zu unterscheiden

Fall 1 ($\alpha[j_{l-k}]$ existiert): Unter der Voraussetzung (*), und da $\alpha[j_l]E_a\alpha[j_{l-k+1}]$ gilt, folgt in diesem Falle (vgl. Hilfssatz 5.2.1/1) $t(\alpha[j_l])\geq t(WP_a(\alpha[j_{l-k+1}]))>t(\alpha[j_{l-k}])$ im Widerspruch zur Minimalität von l, d.h. Fall 1 kann nicht auftreten.

Fall 2 ($\alpha[j_{l-k}]$ existiert nicht): In diesem Falle muß (falls $l=1$ ist trivialerweise, falls $l>1$ ist aufgrund der folgenden Beziehungen) $\alpha[j_l]E_a\alpha[i]$ gelten, denn von

$$\alpha[j_l]E_a^*\alpha[j_{l-k+1}]B_a\alpha[j_{l-k}] \;\&\; proc_P(\alpha[j_l],\alpha[j_{l-k}])$$

kann aufgrund der Beziehung $t(\alpha[j_l])<t(\alpha[j_{l-k}])<t(\alpha[j_{l-k+1}])$ und der Beziehung $\alpha[j_l]E_a\alpha[j_{l-k+1}]$ auf $\alpha[j_l]E_a\alpha[j_{l-k}]$ geschlossen werden ($t(\alpha[j_l])<t(\alpha[j_{l-k}])$ kann aufgrund der Wahl von l gefolgert werden) $\alpha[j_l]E_a\alpha[i]$ folgt schließlich mit vollständiger Induktion
Mit $\alpha[j_l]E_a\alpha[i]$ folgt jedoch $t(\alpha[j_l])\geq t(WP_a(\alpha[i]))$, was einen Widerspruch zur Annahme $t(\alpha[j_l])<t(WP_a(\alpha[i]))$ darstellt. ■

Satz 5.2.2/3: Sei $P=(A,D,\mathcal{W})$ ein (auf einem Monoprozessor ablaufendes) Prozeßsystem. Für jede aktive Berechnung α von P gelte entweder:

$$\forall\alpha[i],\alpha[j],\alpha[k]\in W(\alpha):\ [(\alpha[k]E_a^*\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow\ t(\alpha[k])>t(\alpha[i])]$$

oder

$$\forall\alpha[i],\alpha[j]\in W(\alpha):\ [(\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow\ t(WP_a(\alpha[j]))>t(\alpha[i])].$$

Des weiteren gebe es eine Konstante $n\in\mathbf{N}$ und in bezug auf als fehlerhaft erkannte Aktionen $\alpha[j]\in W(\alpha)$ einer beliebigen Berechnung α gelte:

$$|Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i])\ |\ [t(WP_a(\alpha[j])),t(\alpha[|\alpha|])])|<n.$$

Unter diesen Voraussetzungen ist P dominoeffektfrei, d.h. für jede beliebige aktive Berechnung α von P gilt:

$$|Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))|<n.$$

Beweis: Unter den gemachten Annahmen folgt die Aussage unmittelbar aufgrund der Sätze 5.2.2/1 und 2. ■

5.3. Beziehungsorientierte Sicherungen

Unabhängig von den bereits vorgestellten Konzepten der Leseaufrufsicherung und der wiederaufsetzpunktgesicherten Prozeßwechsel, läßt sich die Dominoeffektfreiheit eines Prozeßsystems auch dann sicherstellen, wenn Wiederaufsetzpunkte in Abhängigkeit von bestehenden Prozeßbeziehungen, d.h. beziehungsorientiert, gesetzt werden.

Die Charakterisierung dreier bzgl. der Beziehungen zwischen Aktionen einer Berechnung interessanter Situationen und der Nachweis der Wirksamkeit ihrer Sicherung durch Wiederaufsetzpunkte erfolgen nacheinander in den beiden nächsten Abschnitten

In die Kategorie beziehungsorientierter Sicherung mit dem Ziel der Vermeidung/Verhinderung des Dominoeffektes fallen auch die in [AL81] grob skizzierten Überlegungen zur Synchronisation des Setzens der Wiederaufsetzpunkte von Gruppen miteinander in Beziehung stehender Prozesse zum Zwecke der Gewährleistung der Existenz einer gemeinsamen *Wiederaufsetzpunktlinie* (*Recovery-Linie*).
Das Konzept der *Konversation*, vgl hierzu [Ran77], [RT79], zielt in die gleiche Richtung. Bei diesem Konzept warten im Extremfall an einem Informationsaustausch beteiligte Prozesse am Ende einer Kommunikationsphase aufeinander, um einen für sie alle gemeinsamen Wiederaufsetzpunkt einrichten zu können.

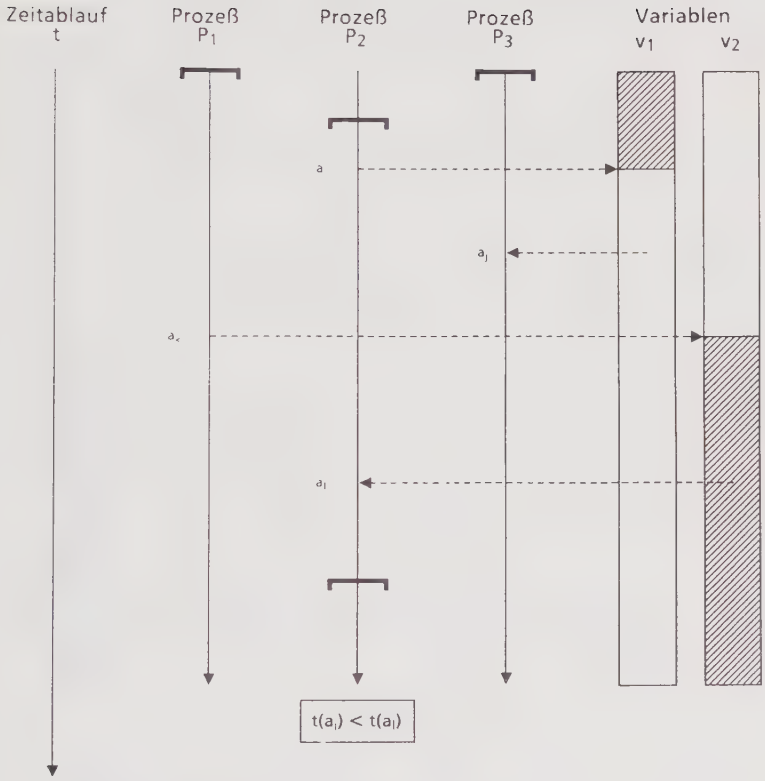
5.3.1. Charakterisierung beziehungsorientierter Sicherungen

Grundvoraussetzung für das Auftreten des Problems der zyklischen Zurücksetzung und des Dominoeffektes ist das Bestehen von Interprozeßbeziehungen. Da sich Interprozeßbeziehungen innerhalb einer aktiven Berechnung vollständig mittels einer Relation B_α beschreiben lassen, liegt es nahe, Beziehungen der Form $\alpha[j]B_\alpha\alpha[i]$ durch Wiederaufsetzpunkte abzusichern. Dies kann beispielsweise dadurch geschehen, in dem auf die Einhaltung der folgenden Forderung geachtet wird:

$$\forall \alpha[i],\alpha[j]\in W(\alpha): [\alpha[j]B_\alpha\alpha[i] \Rightarrow t(WP_\alpha(\alpha[j]))\geq t(WP_\alpha(\alpha[i]))].$$

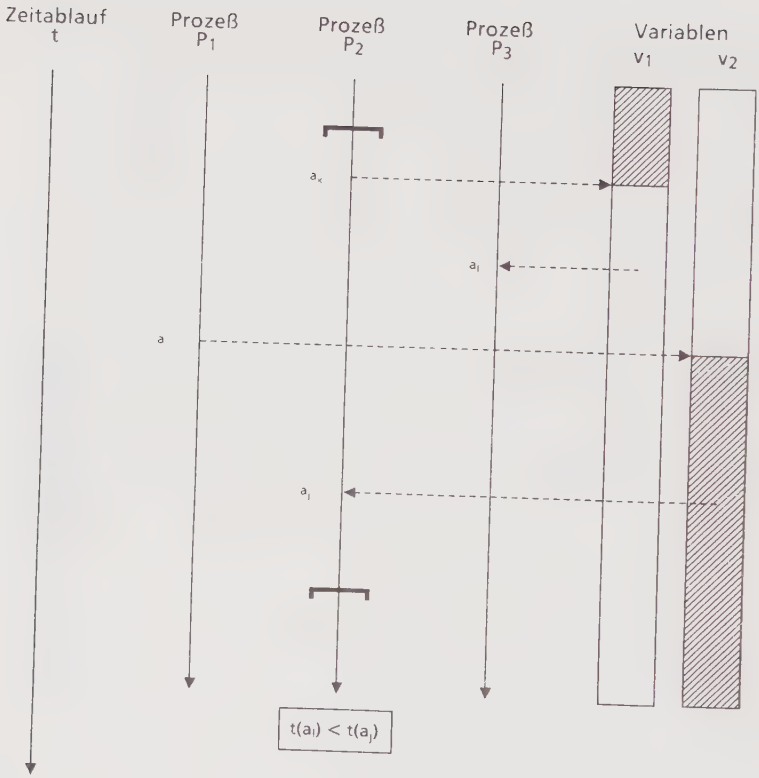
Der Nachweis der Wirksamkeit dieser Forderung erfolgt in den Sätzen 5.3.2/1 und 2.

Eine weitere Möglichkeit besteht in der Absicherung von Situationen, wie sie in den Bildern 5.3.1/1 und 2 skizziert werden und die, für Aktionen $\alpha[i],\alpha[j],\alpha[k],\alpha[l]\in W(\alpha)$ einer aktiven Berechnung α, zu den beiden nachstehend skizzierten einzuhaltenden Bedingungen führen:



Legende: siehe Bild 1.3/1.

Bild 5.3.1/1: Veranschaulichung der Situation bei Nichterfülltsein von Bedingung (*)



Legende: siehe Bild 1.3.1.

Bild 5.3.1/2: Veranschaulichung der Situation bei Nichterfülltsein von Bedingung (**)

$$(*)\ \forall \alpha[i],\alpha[j] \in W(\alpha):\ [(\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow$$
$$(\neg\exists \alpha[k],\alpha[l] \in W(\alpha):\ [\alpha[l]B_a\alpha[k]\ \&\ \neg proc_P(\alpha[k],\alpha[l])\ \&$$
$$\alpha[l]E_a\alpha[i]\ \&\ t(\alpha[l])>t(\alpha[i])])]$$

und

$$(**)\ \forall \alpha[i],\alpha[j] \in W(\alpha):\ [(\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow$$
$$(\neg\exists \alpha[k],\alpha[l] \in W(\alpha)\ [\alpha[l]B_a\alpha[k]\ \&\ \neg proc_P(\alpha[k],\alpha[l])\ \&$$
$$\alpha[k]E_a\alpha[j]\ \&\ t(\alpha[l])<t(\alpha[j])])].$$

Die Aussage des im folgenden bewiesenen Hilfssatzes 5.3.1/1 wird zeigen, daß Bedingung (**) erfüllt ist, wenn (*) Gültigkeit besitzt. Aus diesem Grunde kann vom Nachweis der Wirksamkeit von Bedingung (**) in den Satzen 5.3.2/3 und 4 auf die Wirksamkeit von (*) geschlossen werden (vgl. Korollar 5.3.2/1 und 2).

Hilfssatz 5.3.1/1: Ein Prozeßsystem $P=(A,D,w)$ erfülle in bezug auf alle Aktionen $\alpha[i],\alpha[j]$ jeder aktiven Berechnungsfolge α von P die Bedingung:

$$(*)\ \forall \alpha[i],\alpha[j] \in W(\alpha):\ [(\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow$$
$$(\neg\exists \alpha[k],\alpha[l] \in W(\alpha):\ [\alpha[l]B_a\alpha[k]\ \&\ \neg proc_P(\alpha[k],\alpha[l])\ \&$$
$$\alpha[l]E_a\alpha[i]\ \&\ t(\alpha[l])>t(\alpha[i])])].$$

Dann erfüllt P auch die Bedingung:

$$(**)\ \forall \alpha[i],\alpha[j] \in W(\alpha):\ [(\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow$$
$$(\neg\exists \alpha[k],\alpha[l] \in W(\alpha):\ [\alpha[l]B_a\alpha[k]\ \&\ \neg proc_P(\alpha[k],\alpha[l])\ \&$$
$$\alpha[k]E_a\alpha[j]\ \&\ t(\alpha[l])<t(\alpha[j])])].$$

Das heißt: (*) => (**)

Beweis: Da "(*)=>(**)" äquivalent ist zu "¬(**)=>¬(*)", genügt es im folgenden "¬(**)=>¬(*)" zu beweisen.
¬(**) bedeutet, daß eine aktive Berechnungsfolge α von P existiert, für die gilt:

$$\exists \alpha[i],\alpha[j],\alpha[k],\alpha[l] \in W(\alpha):\ [\alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j])\ \&\ \alpha[k]E_a\alpha[j]\ \&$$
$$\alpha[l]B_a\alpha[k]\ \&\ \neg proc_P(\alpha[k],\alpha[l])\ \&\ t(\alpha[l])\leq t(\alpha[j])].$$

Da $\alpha[l]B_a\alpha[k]\ \&\ t(\alpha[l])\leq t(\alpha[j])$ gilt, folgt aufgrund der Definition der Relation B_a und der Transivität der Relation "≤":

$$t(\alpha[k]) \leq t(\alpha[j]).$$

Damit gilt:

$$\exists\alpha[i],\alpha[j],\alpha[k],\alpha[l]\in W(\alpha):\ [\alpha[l]B_a\alpha[k]\ \&\ \neg proc_P(\alpha[k],\alpha[l])\ \&\ \alpha[k]E_a\alpha[j]\ \&\ \alpha[j]B_a\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j])\ \&\ t(\alpha[k])\leq t(\alpha[j])],$$

und dies entspricht der Situation $\neg(*)$ bei Umbenennung von k,l,i,j in i,j,k,l. ■

Zum Beweis der Aussagen des nächsten Abschnittes wird, neben Hilfssatz 5.3.1/1, der folgende, sich auf (**) beziehender Hilfssatz benötigt:

Hilfssatz 5.3.1/2: Sei α eine aktive Berechnungsfolge des Prozeßsystems $P=(A,D,\mathfrak{u})$ und für alle $\alpha[i],\alpha[j]\in W(\alpha)$ gelte die Bedingung (**) des Hilfssatzes 5.3.1/1. Dann läßt sich zeigen:

(1) $\alpha[j]=\alpha[j_m](E_a\cup B_a)\alpha[j_{m-1}]\ldots\alpha[j_2](E_a\cup B_a)\alpha[j_1]=\alpha[i]\ \Rightarrow$
$[\forall l\in\{2,\ldots,m\}:\ [(\alpha[j_l]B_a\alpha[j_{l-1}]\ \&\ \neg\alpha[j_l]E_a\alpha[j_{l-1}])\Rightarrow t(\alpha[j_l])\geq t(WP_a(\alpha[i]))]],$

(2) $\forall\alpha[j]\in Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])}F_{nv(a)}(\alpha[i]))\exists\alpha[i]\in W(\alpha):\ [f_a(\alpha[i])=\mathbf{true}\ \&$
$(t(\alpha[j])\geq t(WP_a(\alpha[i]))\ \|$
$\exists\alpha[k]\in Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])}F_{nv(a)}(\alpha[i])):\ [\alpha[k]E_a\alpha[j]\ \&\ t(\alpha[k])\geq t(WP_a(\alpha[i]))])].$

Beweis:

zu(1): Mit Hilfe vollständiger Induktion wird gezeigt, daß unter den Voraussetzungen des Hilfssatzes Aussage (1) gültig ist.

Induktionsanfang $t(\alpha[j_2])\geq t(WP_a(\alpha[i]))$ gilt im Falle $\alpha[j_2]B_a\alpha[i]\ \&\ \neg\alpha[j_2]E_a\alpha[i]$ aufgrund der Definition von B_a.

Induktionsannahme: $\forall l\in\{2,\ldots,n\},\ n<m\ \ [(\alpha[j_l]B_a\alpha[j_{l-1}]\ \&\ \neg\alpha[j_l]E_a\alpha[j_{l-1}])\Rightarrow$
$t(\alpha[j_l])\geq t(WP_a(\alpha[i]))].$

Induktionsschluß: Es ist zu zeigen:

$$(\alpha[j_{n+1}]B_a\alpha[j_n]\ \&\ \neg\alpha[j_{n+1}]E_a\alpha[j_n])\ \Rightarrow\ t(\alpha[j_{n+1}])\geq t(WP_a(\alpha[i]))$$

Wähle maximales k, $0\leq k<n$, mit der Eigenschaft $\alpha[j_n]E_a^*\alpha[j_{n-k}]$. Zwei Fälle sind zu unterscheiden.

Fall 1 ($k=n-1$): In diesem Falle gilt $\alpha[j_{n+1}]B_a\alpha[j_n]E_a\alpha[j_1]=\alpha[i]$ und damit $t(\alpha[j_{n+1}])>t(\alpha[j_n])\geq t(WP_a(\alpha[j_n]))=t(WP_a(\alpha[i]))$.

Fall 2 ($k<n-1$)): In diesem Falle gilt:

$$\alpha[j_{n+1}]B_a\alpha[j_n]E_a\alpha[j_{n-k}]B_a\alpha[j_{n-k-1}]\ \&\ \neg\alpha[j_{n-k}]E_a\alpha[j_{n-k-1}]$$

sowie nach Induktionsannahme. $t(\alpha[j_{n-k}])\geq t(WP_a(\alpha[i]))$.
An dieser Stelle gilt es wiederum zwei Fälle zu unterscheiden:

Fall 2.1 $(proc_P(\alpha[j_{n+1}],\alpha[j_n]))\quad t(\alpha[j_{n+1}])>max(t(\alpha[j_n]),t(\alpha[j_{n-k}]))\geq t(WP_a(\alpha[i]))$, denn nach Voraussetzung gilt:

$$\alpha[j_{n+1}]B_a\alpha[j_n]\ \&\ \neg\alpha[j_{n+1}]E_a\alpha[j_n].$$

Fall 2.2 $(\neg proc_P(\alpha[j_{n+1}],\alpha[j_n]))$: Im Falle

$$proc_P(\alpha[j_{n-k}],\alpha[j_{n-k-1}])\ \&\ \alpha[j_{n-k-1}]E_a\alpha[i]$$

gilt

$$t(\alpha[j_{n+1}])>min(t(\alpha[j_n]),t(\alpha[j_{n-k}]))>t(\alpha[j_{n-k-1}])\geq t(WP_a(\alpha[i])).$$

Im Falle $proc_P(\alpha[j_{n-k}],\alpha[j_{n-k-1}])\ \&\ \neg\alpha[j_{n-k-1}]E_a\alpha[i]$ muß ein $\alpha[j_{n-k-1-l}]$ existieren, $\alpha[j_{n-k-1-l}]E_a\alpha[j_{n-k-1}]\ \&\ 0\leq l<n-k-2$, mit der Eigenschaft

$$t(\alpha[j_{n-k-1-l}])\geq t(WP_a(\alpha[i]))$$

und daher kann

$$t(\alpha[j_{n+1}])>min(t(\alpha[j_n]),t(\alpha[j_{n-k}]))>max(t(\alpha[j_{n-k-1}]),t(\alpha[j_{n-k-1-l}]))\geq t(WP_a(\alpha[i]))$$

gefolgert werden.
Im Falle von $\neg proc_P(\alpha[j_{n-k}],\alpha[j_{n-k-1}])$ folgt $t(\alpha[j_{n+1}])>t(\alpha[j_{n-k}])\geq t(WP_a(\alpha[i]))$ aufgrund der Voraussetzung (**)

zu(2): Angenommen (2) würde nicht gelten, d.h. es gäbe ein

$$\alpha[j]\in Z_a\Big(\bigcup_{\alpha[i]\in W(\alpha)\,\&\,f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i])\Big)$$

und für alle

$$\alpha[k]\in Z_a\Big(\bigcup_{\alpha[i]\in W(\alpha)\,\&\,f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i])\Big),\ k\neq j,$$

würde

$$\neg\alpha[j]E_a\alpha[k]\ \|\ t(\alpha[k])<t(WP_a(\alpha[i]))$$

gelten; für ein $\alpha[i]\in W(\alpha)$, $f_a(\alpha[i])=$**true**, insbesondere auch $t(\alpha[j])<t(WP_a(\alpha[i]))$.
Da $\alpha[j]\in Z_a\Big(\bigcup_{\alpha[i]\in W(\alpha)\,\&\,f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i])\Big)$, muß es eine Aktion $\alpha[i]\in W(\alpha)$, $f_a(\alpha[i])=$**true**,

und nach Hilfssatz 3.2.4/5 Aktionen $\alpha[j_l]\in W(\alpha)$, $1\leq l\leq m$, geben, so daß gilt:

$$\alpha[j]=\alpha[j_m](E_a\cup B_a)\alpha[j_{m-1}]\ldots\alpha[j_2](E_a\cup B_a)\alpha[j_1]=\alpha[i].$$

Mit Hilfe der vollständigen Induktion läßt sich zeigen, daß für alle $\alpha[j_l]$, $1<l\leq m$, $\alpha[j_l]E_a\alpha[j_{l-1}]$ gelten müßte.

Induktionsanfang: $\alpha[j_m]E_a\alpha[j_{m-1}]$. Angenommen

$$\alpha[j_m]B_a\alpha[j_{m-1}]\ \&\ \neg\alpha[j_m]E_a\alpha[j_{m-1}]$$

würde gelten. Dann würde aufgrund von (1) $t(\alpha[j_m])\geq t(WP_a(\alpha[i]))$ folgen im Widerspruch zur Annahme $t(\alpha[j])<t(WP_a(\alpha[i]))$.

Induktionsannahme: $\forall l\in\{m,\ldots,m-n+1\}$, $n<m-1$, $\alpha[j_l]E_a\alpha[j_{l-1}]$.

Induktionsschluß: $\alpha[j_{m-n}]E_a\alpha[j_{m-n-1}]$, $n<m-1$. Angenommen

$$\neg\alpha[j_{m-n}]E_a\alpha[j_{m-n-1}]\ \&\ \alpha[j_{m-n}]B_a\alpha[j_{m-n-1}]$$

würde gelten. Dann würde aufgrund von (1) $t(\alpha[j_{m-n}])>t(WP_a(\alpha[i]))$ folgen, und, da nach Induktionsannahme für $m\geq l\geq m-n+1$ $\alpha[j_l]E_a\alpha[j_{l-1}]$ gilt, würde, z.B. bei Wahl von $k:=j_{m-n}$, gelten:

$$\exists\alpha[k]\in W(\alpha):[\alpha[k]E_a\alpha[j]\ \&\ t(\alpha[k])\geq t(WP_a(\alpha[i]))]$$

Dies würde einen Widerspruch zur Annahme darstellen.

Insgesamt ist damit für $1\leq l\leq m$ $\alpha[j_l]E_a\alpha[i]$ und insbesondere $\alpha[j]=\alpha[j_m]E_a\alpha[i]$ gezeigt. $\alpha[j]E_a\alpha[i]$ steht jedoch im Widerspruch zur Annahme:

$$\forall\alpha[k]\in Z_a\Big(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])}F_{nv(\alpha)}(\alpha[i])\Big):[\neg\alpha[j]E_a\alpha[k]\ \|\ t(\alpha[k])<t(WP_a(\alpha[i]))],$$

denn in diesem Fall existiert ein $\alpha[k]\in Z_a\Big(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])}F_{nv(\alpha)}(\alpha[i])\Big)$ mit der Eigenschaft $t(\alpha[k])\geq t(WP_a(\alpha[i]))$, nämlich $\alpha[i]$ selbst. ■

5.3.2. Wirksamkeit beziehungsorientierter Sicherungen

Im Verlauf dieses Abschnittes wird nun die Wirksamkeit beziehungsorientierter Sicherungen, wie sie im vorangehenden Abschnitt motiviert wurden, im Rahmen der Sätze 5.3.2/1 bis 4 und zwei auf den Sätzen 5.3.2/3 und 4 bzw. auf Hilfssatz 5.3.1/1 basierender Schlußfolgerungen (Korollar 5.3.2/1 und 2) gezeigt.

Satz 5.3.2/1: Sei $P=(A,D,w)$ ein Prozeßsystem und α eine aktive Berechnung von P. Für alle Aktionen $\alpha[i],\alpha[j]\in W(\alpha)$ gelte

$$(*)\ \alpha[j]B_a\alpha[i] \Rightarrow t(WP_a(\alpha[j]))\geq t(WP_a(\alpha[i])).$$

In diesem Falle läßt sich zeigen, daß die Zurücksetzung einer Aktion $\alpha[i]\in W(\alpha)$ nur die Zurücksetzung von Aktionen $\alpha[j]\in W(\alpha)$, die nach $WP_a(\alpha[i])$ ausgeführt wurden, d.h. für die $t(\alpha[j])\geq t(WP_a(\alpha[i]))$ gilt, erforderlich machen kann.

Beweis: Für jede Aktion $\alpha[j]$, die bei Zurücksetzung von $\alpha[i]$ zurückgesetzt werden muß, gilt $\alpha[j](E_a\cup B_a)^*\alpha[i]$. Daher muß es nach Hilfssatz 3.2.4/5 Aktionen $\alpha[j_1],\ldots,\alpha[j_m]\in W(\alpha)$ geben mit der Eigenschaft:

$$\alpha[j]=\alpha[j_m](E_a\cup B_a)\alpha[j_{m-1}]\ldots\alpha[j_2](E_a\cup B_a)\alpha[j_1]=\alpha[i].$$

Zum Beweis des Satzes genügt es zu zeigen, daß für alle $l\in\{1,\ldots,m\}$ gilt:

$$t(WP_a(\alpha[j_l]))\geq t(WP_a(\alpha[i])).$$

Dies wird mit Hilfe vollständiger Induktion nachgewiesen.

Induktionsanfang $(t(WP_a(\alpha[j_1]))\geq t(WP_a(\alpha[i])))$: Zwei Fälle sind zu unterscheiden:

Fall 1 $(\alpha[j_1]E_a\alpha[i])$ $t(WP_a(\alpha[j_1]))\geq t(WP_a(\alpha[i]))$ folgt mit der Definition der Relation E_a.

Fall 2 $(\alpha[j_1]B_a\alpha[i])$ $t(WP_a(\alpha[j_1]))\geq t(WP_a(\alpha[i]))$ kann aufgrund der Voraussetzung $(*)$ gefolgert werden.

Induktionsannahme: $(t(WP_a(\alpha[j_n]))\geq t(WP_a(\alpha[i]))),\ 1\leq n<m$.

Induktionsschluß $(t(WP_a(\alpha[j_{n+1}]))\geq t(WP_a(\alpha[i])),\ 1\leq n<m)$: Zwei Fälle sind zu unterscheiden:

Fall 1 $(\alpha[j_{n+1}]E_a\alpha[j_n])$ $t(WP_a(\alpha[j_{n+1}]))\geq t(WP_a(\alpha[j_n]))$ folgt nach Definition der Relation E_a.

Fall 2 $(\alpha[j_{n+1}]B_a\alpha[j_n])$: $t(WP_a(\alpha[j_{n+1}]))\geq t(WP_a(\alpha[j_n]))$ folgt aufgrund der Voraussetzung $(*)$.

Da in beiden Fällen $t(WP_a(\alpha[j_{n+1}]))\geq t(WP_a(\alpha[j_n]))$ und nach Induktionsannahme $t(WP_a(\alpha[j_n]))\geq t(WP_a(\alpha[i]))$ gilt, kann auf $t(WP_a(\alpha[j_{n+1}]))\geq t(WP_a(\alpha[i]))$ geschlossen werden. ■

Satz 5.3.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem. Für jede aktive Berechnung α von P

gelte

$$\forall \alpha[i],\alpha[j]\in W(\alpha)\colon [\alpha[j]B_a\alpha[i] \Rightarrow t(WP_a(\alpha[j]))\geq t(WP_a(\alpha[i]))].$$

Des weiteren gebe es eine Konstante $n\in\mathbb{N}$, und in bezug auf als fehlerhaft erkannte Aktionen $\alpha[j]\in W(\alpha)$ gelte:

$$|Z_a(\bigcup_{\alpha[i]\in W(\alpha)\,\&\,f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(WP_a(\alpha[j])),t(\alpha[|\alpha|])])|<n.$$

Unter diesen Voraussetzungen ist P dominoeffektfrei, d.h. für jede beliebige aktive Berechnung α von P gilt:

$$|Z_a(\bigcup_{\alpha[i]\in W(\alpha)\,\&\,f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))|<n.$$

Beweis: Unter Zugrundelegung der Voraussetzungen folgt die Aussage unmittelbar mit Satz 5.3.2/1. ■

Satz 5.3.2/3: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1,\ldots,P_n$, und für den in P aktiven Prozeßablauf α gelte die Bedingung (**) des Hilfssatzes 5.3.1/1, d.h.:

$$\forall \alpha[i],\alpha[j]\in W(\alpha)\colon [(\alpha[j]B_a\alpha[i] \;\&\; \neg proc_P(\alpha[i],\alpha[j])) \Rightarrow (\neg\exists \alpha[k],\alpha[l]\in W(\alpha)\colon [\alpha[l]B_a\alpha[k] \;\&\; \neg proc_P(\alpha[k],\alpha[l]) \;\&\; \alpha[k]E_a\alpha[j] \;\&\; t(\alpha[l])<t(\alpha[j]))]].$$

Des weiteren existiere eine *maximale Recoveryeinheitsgröße* $g_{re}\in\mathbb{N}$, so daß für alle aktiven Berechnungen α von P gilt:

$$g_{re_a} ::= \max_{\alpha[k]\in W(\alpha)} |\{\alpha[j] \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[j]E_a\alpha[k]\}| < g_{re}.$$

Dann kann die Zurücksetzung einer Aktion $\alpha[i]\in W(\alpha)$ die Zurücksetzung von maximal m bzgl. des Indexes j_l, $1\leq l\leq m$, verschiedenen Aktionen $\alpha[j_1],\ldots,\alpha[j_m]$, $m<g_{re}-1$, mit der Eigenschaft $t(\alpha[j_l])<t(WP_a(\alpha[i]))$, $1\leq l\leq m$, eines jeden Prozesses P_k, $1\leq k\leq n$, von P erfordern.

Beweis: Angenommen die Zurücksetzung einer Aktion $\alpha[i]$ würde die Zurücksetzung von $m\geq g_{re}-1$ Aktionen $\alpha[j_1],\ldots,\alpha[j_m]$ mit der Eigenschaft $t(\alpha[j_l])<t(WP_a(\alpha[i]))$ eines Prozesses P_k von P erforderlich machen.
Nach Hilfssatz 5.3.1/2-(2) muß zu jedem $\alpha[j_l]$ ein $\alpha[k_l]$ existieren mit der Eigenschaft $\alpha[j_l]E_a\alpha[k_l]$ & $t(\alpha[k_l])\geq t(WP_a(\alpha[i]))$.
Sei $q\in\{1,\ldots,m\}$ so gewählt, daß $t(\alpha[j_q])=min\{t(\alpha[j_l]) \mid 1\leq l\leq m\}$. Da $\alpha[j_q]E_a\alpha[k_q]$, d.h. $WP_a(\alpha[j_q])=WP_a(\alpha[k_q])$ gilt, die $\alpha[j_l]$, $1\leq l\leq m$, alle zum gleichen Prozeß gehören und $t(\alpha[j_l])\in[t(\alpha[j_q]),t(\alpha[k_q])[$ gilt, folgt, daß $WP_a(\alpha[j_l])=WP_a(\alpha[k_q])$ gilt, und somit $m+1\geq g_{re}$ Aktionen $\alpha[j_1],\ldots,\alpha[j_m],\alpha[k_q]$ zur gleichen Recoveryeinheit gehören. Dies steht im Widerspruch zur Annahme, daß die maximale Recoveryeinheitsgröße von α

kleiner g_{re} ist ■

Korollar 5.3.2/1: Unter den Voraussetzungen des vorangehenden Satzes 5.3.2/3 kann die Zurücksetzung einer Aktion $\alpha[i]\in W(\alpha)$ lediglich die Zurücksetzung von weniger als $n^*(g_{re}-1)$ Aktionen $\alpha[j]$ mit der Eigenschaft $t(\alpha[j])<t(WP_\alpha(\alpha[i]))$ erfordern.

Beweis: Folgt unmittelbar mit Satz 5.3.2/3 und der Annahme das Prozeßsystem P bestehe aus n Prozessen. ■

Satz 5.3.2/4: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1,...,P_n$, und für jede aktive Berechnungsfolge α gelte die Bedingung (**) des Hilfssatzes 5.3.1/1, d.h.:

$$\forall\alpha[i],\alpha[j]\in W(\alpha):\ [(\alpha[j]B_\alpha\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j]))\ \Rightarrow\ (\neg\exists\alpha[k],\alpha[l]\in W(\alpha):\ [\alpha[l]B_\alpha\alpha[k]\ \&\ \neg proc_P(\alpha[k],\alpha[l])\ \&\ \alpha[k]E_\alpha\alpha[j]\ \&\ t(\alpha[l])<t(\alpha[j])])].$$

Des weiteren gebe es Konstanten $n_1,n_2\in\mathbf{N}$, so daß für jede aktive Berechnung α gelte:

(1) $\forall\alpha[j]\in W(\alpha):\ [|\{i\mid\alpha[i]\in W(\alpha)\ \&\ \alpha[i]E_\alpha\alpha[j]\}|<n_1]$ und

(2) in bezug auf alle als fehlerhaft erkannten Aktionen $\alpha[j]\in W(\alpha)$:

$$|Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{nv(\alpha)}(\alpha[i])\mid[t(WP_\alpha(\alpha[j])),t(\alpha[|\alpha|])])|<n_2.$$

Unter diesen Voraussetzungen ist P dominoeffektfrei, d.h. für jede beliebige aktive Berechnung α von P gilt:

$$|Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{nv(\alpha)}(\alpha[i]))|<n^*(n_1-1)+n_2.$$

Beweis: Zunächst läßt sich die Größe der Menge $Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{nv(\alpha)}(\alpha[i]))$ wie folgt abschätzen:

$$|Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{nv(\alpha)}(\alpha[i]))|\le|Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{nv(\alpha)}(\alpha[i])\mid[t(\alpha[1]),t(WP_\alpha(\alpha[i]))[)|+$$
$$|Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{nv(\alpha)}(\alpha[i])\mid[t(WP_\alpha(\alpha[i])),t(\alpha[|\alpha|])])|<$$
$$|Z_\alpha(\bigcup_{\alpha[i]\in W(\alpha)\&f_\alpha(\alpha[i])}F_{nv(\alpha)}(\alpha[i])\mid[t(\alpha[1]),t(WP_\alpha(\alpha[i]))[)|+n_2.$$

Bleibt zu zeigen, daß:

$$|Z_a(\bigcup_{\alpha[i]\in W(a)\&f_a(\alpha[i])} F_{nw(a)}(\alpha[i])\,|\,[t(\alpha[1]),t(WP_a(\alpha[i]))[])|<n^*(n_1-1)$$

ist. Dies folgt unmittelbar mit Korollar 5.3.2/1. ■

Korollar 5.3.2/2: Die Aussage von Korollar 5.3.2/1 und des Satzes 5.3.2/4 bleibt bei Ersetzung der Bedingung (**) des Hilfssatzes 5.3.1/1 durch die Bedingung (*) des Hilfssatzes 5.3.1/1 gültig.

Beweis: Folgt mit Hilssatz 5.3.1/1, in dem (*) => (**) nachgewiesen wurde. ■

5.4. Transaktionsorientierte Sicherungen

Eine weitere Möglichkeit Dominoeffektfreiheit von Prozeßsystemen zu erreichen, besteht in der Reglementierung des Zugriffs auf Variablenwerte innerhalb einer Recoveryeinheit durch ein Protokoll. Beispielsweise können zwischen Recoveryeinheiten ausschließlich transaktionsorientierte Beziehungen für zulässig erklärt werden, wobei eine *Transaktion*, vgl. [Reu81], [Wed76], [Wed81], als eine Recoveryeinheit mit speziellen Eigenschaften zu modellieren ist. Eine dieser Eigenschaften kann dabei der *Grad der Konsistenzerhaltung* einer Transaktion, vgl. [Gra75], sein.

In der Literatur werden im Zusammenhang mit dem *Transaktionsbegriff* verschiedene *Konsistenzgrade* unterschieden.
Eine *Transaktion T* wird z.B. in [Gra75] genau dann als den *Konsistenzgrad i erhaltend*, $0 \leq i \leq 3$, bezeichnet, wenn sie den folgenden Kriterien (0) bis (3) genügt:

(0) *T* überschreibt von anderen Transaktionen neu erzeugte oder veränderte Daten so lange nicht, bis sie von diesen freigegeben werden (*"T does not overwrite dirty, i.e. uncommitted, data of other transactions"*).
Erläuterung: Eine Transaktion kann ein Datum freigeben, wenn es von ihr nicht mehr verändert wird.

(1) *T* gibt neu erzeugte oder veränderte Daten auf einmal (am Transaktionsende) frei (*"T does not commit any writes before the end of transaction (EOT)"*).

(2) *T* liest keine durch andere Transaktionen neu erzeugten oder veränderten Daten, bevor ihre Freigabe nicht erfolgt ist (*"T does not read dirty data of other transactions"*).

(3) Von *T* gelesene Daten werden von anderen Transaktionen nicht vor dem Transaktionsende von *T* verändert (*"Other transactions do not dirty any data read by T before T completes"*).

In [Gra75] werden Transaktionen des weiteren in zwei Klassen unterteilt: In die *Klasse der ohne Auswirkungen auf andere rücksetzbaren Transaktionen* und die *Klasse der nicht ohne (die Gefahr der) Rücksetzung anderer (Dominoeffekt möglich) rücksetzbaren Transaktionen*.
Es wird behauptet, daß:

(i) Konsistenzgrad 0 erhaltende Transaktionen nicht ohne die Gefahr der Rücksetzung anderer Transaktionen rücksetzbar sind,

(ii) wenn alle Transaktionen eines Systems Konsistenzgrad 0 erhaltend sind, dann ist jede Transaktion, die Konsistenzgrad 1 erhaltend ist, ohne Auswirkungen auf andere Transaktionen rücksetzbar

Behauptung (i) ist nur unter zwei Annahmen richtig:

(a) Eine Konsistenzgrad 1 erhaltende Transaktion darf nach erfolgreichem Abschluß nicht zurückgesetzt werden (bzw. nach der Freigabe der von ihr veränderten oder neu erzeugten Daten nicht zurückgesetzt werden),

(b) die Rücksetzung einer Transaktion bleibt ohne Auswirkungen auf diejenigen Transaktionen, die nicht freigegebene Daten der zurückzusetzenden Transaktion gelesen haben,

und es stellt sich deshalb die Frage, ob unter schwächeren Annahmen ein ähnliches Ergebnis in dem vorliegenden Prozeßsystem- und Fehlerbehandlungs-Modell abgeleitet werden kann.

Eine Antwort auf die zuletzt aufgeworfene Frage zu geben, wird im folgenden versucht.
Zunächst wird skizziert, daß auf die Forderung nach der konsistenzerhaltenden Eigenschaft (0) einer Transaktion bei Verwendung einer entsprechenden Recoverystrategie verzichtet werden kann. Danach wird verdeutlicht, daß bei Wegfall der Annahmen (a)&(b) weder die Eigenschaften (0)&(1)&(3) noch die Eigenschaften (0)&(2)&(3) alleine ausreichen, um den Dominoeffekt im Fehlerfall zu verhindern. Schließlich wird gezeigt, daß die konsistenzerhaltenden Eigenschaften (1)&(2) (genauer: ihre Einhaltung) genügen (genügt), um in bezug auf die Dominoeffektfreiheit eines Prozeßsystems $P=(A,D,w)$ interessante Ergebnisse beweisen zu können.
Voraussetzung für die folgende Diskussion, insbesondere den letzten Punkt, ist die Präzisierung der konsistenzerhaltenden Eigenschaften, wie in den Punkten (0) bis (3) beschrieben. Sie wird daher vorgezogen und vorab durchgeführt.

5.4.1. Charakterisierung transaktionsorientierter Sicherungen

Im für die Untersuchung des Dominoeffektes in Prozeßsystemen gewählten Kontext entspricht eine Transaktion einer aus mehreren atomaren Aktionen bestehenden Recoveryeinheit, und die Punkte (0) bis (3) lassen sich hier in bezug auf eine aktive Berechnungsfolge α eines Prozeßsystems P wie folgt präzisieren ($\alpha[j]$ ist im folgenden als eine atomare Aktion der Transaktion T, vgl. (0) bis (3), zu sehen; des weiteren ist

"(i)=>: ..." zu lesen als: "Ist (i) erfüllt/gültig für alle Transaktionen eines Systems, dann folgt ..."

(0) =>

$$\forall \alpha[i],\alpha[j] \in W(\alpha): [(SV_{\alpha[i]} \cap SV_{\alpha[j]} \neq \phi \;\&\; t(\alpha[i]) < t(\alpha[j]) \;\&\; \neg \alpha[i] E_a \alpha[j]) \Rightarrow \forall \alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha) \;\&\; \alpha[l] E_a \alpha[i] \;\&\; SV_{\alpha[l]} \cap SV_{\alpha[i]} \cap SV_{\alpha[j]} \neq \phi\}: [t(\alpha[k]) < t(\alpha[j])]];$$

(0)&(1) =>:

$$\forall \alpha[i],\alpha[j] \in W(\alpha): [(SV_{\alpha[i]} \cap SV_{\alpha[j]} \neq \phi \;\&\; t(\alpha[j]) < t(\alpha[i]) \;\&\; \neg \alpha[i] E_a \alpha[j]) \Rightarrow \forall \alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha) \;\&\; \alpha[l] E_a \alpha[j]\; [[\& \; SV_{\alpha[l]} \cap gV_P \neq \phi]]\}: [t(\alpha[k]) < t(\alpha[i])]];$$

Der Zusatz "$[[\& \; SV_{\alpha[l]} \cap gV_P \neq \phi]]$" kann interpretiert werden als: Transaktion T gibt alle von ihr neu erzeugten oder veränderten globalen Daten auf einmal, aber nicht unbedingt am Transaktionsende, frei.

(2) =>:

$$\forall \alpha[i],\alpha[j] \in W(\alpha): [(\alpha[j] B_a \alpha[i] \;\&\; \neg \alpha[j] E_a \alpha[i]) \Rightarrow \forall \alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha) \;\&\; \alpha[l] E_a \alpha[i] \;\&\; SV_{\alpha[l]} \cap SV_{\alpha[i]} \cap LV_{\alpha[j]} \neq \phi\}: [t(\alpha[k]) < t(\alpha[j])]];$$

(1)&(2) =>:
entspricht (2) ohne den Zusatz "$SV_{\alpha[l]} \cap SV_{\alpha[i]} \cap LV_{\alpha[j]} \neq \phi$";

(0)&(1)&(2) =>:
entspricht (1)&(2) mit dem Zusatz "$SV_{\alpha[i]} \cap V_{\alpha[j]} \neq \phi \;\&\; t(\alpha[i]) < t(\alpha[j])$" statt "$\alpha[j] B_a \alpha[i]$";

(3) =>:

$$\forall \alpha[i],\alpha[j] \in W(\alpha): [(LV_{\alpha[j]} \cap SV_{\alpha[i]} \;\&\; t(\alpha[j]) < t(\alpha[i]) \;\&\; \neg \alpha[j] E_a \alpha[i]) \Rightarrow \forall \alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha) \;\&\; \alpha[l] E_a \alpha[j]\; [[\& \; LV_{\alpha[l]} \cap gV_P \neq \phi]\; [\& \; LV_{\alpha[l]} \cap LV_{\alpha[j]} \cap SV_{\alpha[i]} \neq \phi]]\}: [t(\alpha[k]) < t(\alpha[i])]];$$

Der Zusatz "$[\& \; LV_{\alpha[l]} \cap gV_P \neq \phi]$" kann interpretiert werden als: Von T gelesene Daten werden von anderen Transaktionen nicht verändert, bevor T nicht alle Leseaufrufe abgeschlossen hat.
Der Zusatz "$[\& \; LV_{\alpha[l]} \cap LV_{\alpha[j]} \cap SV_{\alpha[i]} \neq \phi]$" kann interpretiert werden als: Von T gelesene Daten werden von anderen Transaktionen nicht verändert, so lange sie noch von T gelesen werden.

Nach der kurzen Präzisierung der Eigenschaften (0) bis (3) wird nun erläutert, warum auf die Forderung nach der konsistenzerhaltenden Eigenschaft (0) in einem Transaktionssystem bei Zugrundelegung einer entsprechenden Fehlerbehandlungs-Strategie verzichtet werden kann.

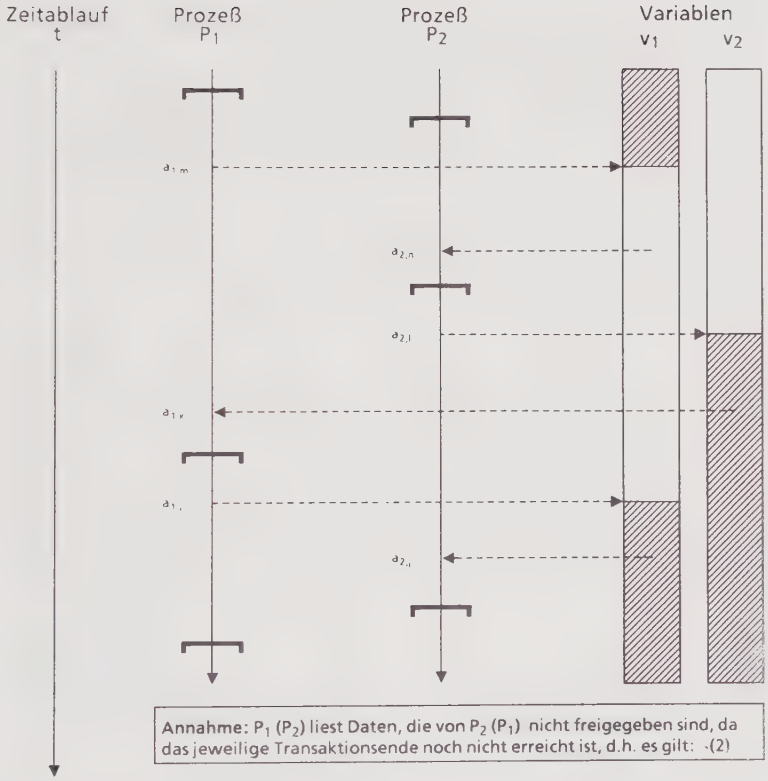
Um die Möglichkeit des Verzichts auf die Forderung nach Eigenschaft (0) zu untermauern, sei angenommen die Transaktionen T_1 und T_2 verändern die Variable v, mit T_1 beginnend. Dann bedeutet die Forderung (0) in bezug auf T_2, daß T_2 v erst dann verändern darf, wenn T_1 alle Änderungen bzgl. v abgeschlossen und v freigegeben hat. Dadurch soll verhindert werden, daß bei Abbruch von T_1 die Zurücksetzung von T_2 erforderlich wird, da bei Abbruch von T_1 Änderungen von T_2 rückgängig gemacht werden. Diese Art von Abhängigkeit kann aber nur dann entstehen, wenn die Änderungen von T_2 bzgl. v im Recoveryfall nicht rekonstruierbar sind. Durch Aufzeichnen sämtlicher Änderungen bzgl. einer Variablen v kann die Rekonstruierbarkeit von v sichergestellt werden.

Nach dem kurz skizziert wurde, warum auf die Forderung (0) verzichtet werden kann, wird nun anhand eines Beispiels gezeigt, daß die Einhaltung der Eigenschaften (0), (1) und (3) in bezug auf die Recoveryeinheiten (Transaktionen) eines Prozeßsystems die Dominoeffektfreiheit eines Prozeßsystems $P=(A,D,w)$ nicht garantiert (vgl. Bild 5.4.1/1).

Obwohl die Prozesse P_1 und P_2 die Forderung (0)&(1)&(3) erfüllen, kann es in der Situation des Bildes 5.4.1/1 über die Aktionen $a_{1,i}, a_{2,j}, a_{1,k}, a_{2,l}, a_{1,m}, a_{2,n}, \ldots$ zum Dominoeffekt kommen.

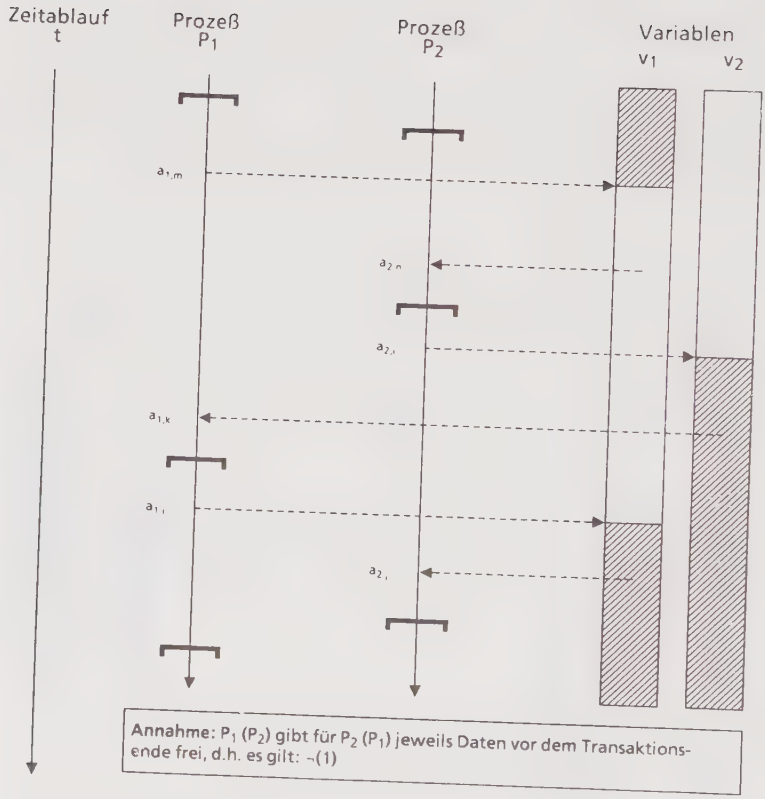
Daß auch die Eigenschaften (0), (2) und (3) die Dominoeffektfreiheit eines Prozeßsystems nicht garantieren können, zeigt das nächste Bild 5.4.1/2.

Obwohl in der Situation des Bildes 5.4.1/2 die Prozesse P_1, P_2 und P_3 die Forderungen (0)&(2)&(3) erfüllen, kann die Zurücksetzung von $a_{1,i}$ die Zurücksetzung unbeschränkbar vieler Aktionen erfordern.



Legende: siehe Bild 1 3/1,

Bild 5.4.1/1: Situation, in der Forderungen (0)&(1)&(3) eingehalten werden können



Legende: siehe Bild 1.3/1.

Bild 5.4.1/2: Situation, in der Forderungen (0)&(2)&(3) eingehalten werden können

5.4.2. Wirksamkeit transaktionsorientierter Sicherungen

Daß prinzipiell die Einhaltung der konsistenzerhaltenden Eigenschaften (1)&(2) ausreicht, um in bezug auf die Dominoeffektfreiheit eines Prozeßsystems interessante Ergebnisse beweisen zu können, wird im folgenden nachgewiesen.
Es wird gezeigt, daß die Einhaltung der Eigenschaften (1)&(2) ausreicht, um eine Abschätzung des Rücksetzaufwandes im Fehlerfall zu ermöglichen. Die Ableitung dieses Ergebnisses erfolgt über mehrere Zwischenschritte.

Hilfssatz 5.4.2/1: Sei α eine aktive Berechnungsfolge des Prozeßsystems $P=(A,D,w)$ und für alle $\alpha[i],\alpha[j]\in W(\alpha)$ gelte (die nachfolgende Forderung (*) entspricht (1)&(2)):

$$(*)\ \ (\alpha[j]B_a\alpha[i]\ \ \&\ \ \neg\alpha[j]E_a\alpha[i]) \Rightarrow (\forall\alpha[k]\in\{\alpha[l]\mid\alpha[l]\in W(\alpha)\ \&\ \alpha[l]E_a\alpha[i]\}:\ [t(\alpha[k])<t(\alpha[j])]).$$

Dann läßt sich zeigen

$$(1)\ \ \alpha[j]=\alpha[j_m](E_a\cup B_a)\alpha[j_{m-1}]\ldots\alpha[j_2](E_a\cup B_a)\alpha[j_1]=\alpha[i] \Rightarrow [\forall l\in\{2,\ldots,m\}\ [(\alpha[j_l]B_a\alpha[j_{l-1}]\ \&\ \neg\alpha[j_l]E_a\alpha[j_{l-1}]) \Rightarrow (\forall k\in\{2,\ldots,l-1\}:\ [t(\alpha[j_l])>t(\alpha[j_{l-k}])])]],$$

$$(2)\ \ \forall\alpha[j]\in Z_a\Big(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i])\Big)\exists\alpha[i]\in W(\alpha):\ [f_a(\alpha[i])=\textbf{true}\ \&\ (t(\alpha[j])\geq t(\alpha[i]))\ \|\ \exists\alpha[l]\in Z_a\Big(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i])\Big)\ [\alpha[l]E_a\alpha[j]\ \&\ t(\alpha[l])\geq t(\alpha[i]))]$$

Beweis:

zu(1): Mit Hilfe vollständiger Induktion wird gezeigt, daß unter den Voraussetzungen des Hilfssatzes Aussage (1) gültig ist.

Induktionsanfang: $t(\alpha[j_l])>t(\alpha[j_{l-1}])$, da nach Voraussetzung $\alpha[j_l]B_a\alpha[j_{l-1}]$ und damit $t(\alpha[j_l])>t(\alpha[j_{l-1}])$ nach Definition von B_a gilt.

Induktionsannahme: $\forall k\in\{2,\ldots,n\}$, $n<l-1$, $t(\alpha[j_l])>t(\alpha[j_{l-k}])$.

Induktionsschluß: $t(\alpha[j_l])>t(\alpha[j_{l-n-1}])$, $n<l-1$. Zwei Fälle sind zu unterscheiden:

Fall 1 $(\alpha[j_{l-n}]B_a\alpha[j_{l-n-1}])$: In diesem Fall ist $t(\alpha[j_{l-n}])>t(\alpha[j_{l-n-1}])$ nach Definition von B_a und die Behauptung folgt mit der Induktionsannahme.

Fall 2 ($\alpha[j_{l-n}]E_a\alpha[j_{l-n-1}]$): Wähle maximales i, $0\leq i<n$, mit der Eigenschaft $\alpha[j_{l-n-1}]E_a\alpha[j_{l-n+i}]$. Da i maximal gewählt wurde und (nach Voraussetzung) $\alpha[j_{l-n+i+1}](E_a\cup B_a)\alpha[j_{l-n+i}]$ gilt, muß

$$\neg\alpha[j_{l-n+i+1}]E_a\alpha[j_{l-n+i}] \text{ und } \alpha[j_{l-n+i+1}]B_a\alpha[j_{l-n+i}]$$

gelten. Aufgrund von (*) folgt daher $t(\alpha[j_{l-n+i+1}])>t(\alpha[j_{l-n-1}])$ und, da nach Induktionsannahme $t(\alpha[j_l])>t(\alpha[j_{l-n+i+1}])$ gilt, folgt die Behauptung auch im Fall 2.

zu(2): Angenommen (2) würde nicht gelten, d.h. es gäbe ein

$$\alpha[j]\in Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))$$

und für alle

$$\alpha[k]\in Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{nw(\alpha)}(\alpha[i]))$$

würde gelten:

$$\neg\alpha[j]E_a\alpha[k] \parallel t(\alpha[k])<t(\alpha[i]);$$

für ein $\alpha[i]\in W(\alpha)$, $f_a(\alpha[i])=$**true**, insbesondere auch $t(\alpha[j])<t(\alpha[i])$.

Da $\alpha[j]\in Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{nw(\alpha)}(\alpha[i]))$, muß es eine Aktion $\alpha[i]\in W(\alpha)$, $f_a(\alpha[i])=$**true**, und nach Hilfssatz 3.2.4/5 Aktionen $\alpha[j_l]\in W(\alpha)$ geben, so daß gilt:

$$\alpha[j]=\alpha[j_m](E_a\cup B_a)\alpha[j_{m-1}]\ldots\alpha[j_2](E_a\cup B_a)\alpha[j_1]=\alpha[i].$$

Mit Hilfe vollständiger Induktion läßt sich zeigen, daß für alle $\alpha[j_l]$, $1<l\leq m$, $\alpha[j_l]E_a\alpha[j_{l-1}]$ gelten müßte.

Induktionsanfang: $\alpha[j_m]E_a\alpha[j_{m-1}]$. Angenommen

$$\alpha[j_m]B_a\alpha[j_{m-1}] \;\&\; \neg\alpha[j_m]E_a\alpha[j_{m-1}]$$

würde gelten. Dann würde aufgrund von (1) $t(\alpha[j_m])>t(\alpha[i])$ folgen im Widerspruch zur Annahme $t(\alpha[j])<t(\alpha[i])$.

Induktionsannahme: $\forall l\in\{m,\ldots,m-n+1\}$, $n<m$, $\alpha[j_l]E_a\alpha[j_{l-1}]$.

Induktionsschluß: $\alpha[j_{m-n}]E_a\alpha[j_{m-n-1}]$, $n<m$. Angenommen

$$\neg\alpha[j_{m-n}]E_a\alpha[j_{m-n-1}] \;\&\; \alpha[j_{m-n}]B_a\alpha[j_{m-n-1}]$$

würde gelten. Dann würde aufgrund von (1) $t(\alpha[j_{m-n}])>t(\alpha[i])$ folgen, und, da nach Induktionsannahme für $m\leq l<m-n+1$ $\alpha[j_l]E_a\alpha[j_{l-1}]$ gilt, würde, z.B. bei Wahl von $k:=j_{m-n}$, gelten:

$$\exists\alpha[k]\colon [\alpha[k]E_a\alpha[j] \;\&\; t(\alpha[k])\geq t(\alpha[i])].$$

Dies wäre ein Widerspruch zur Annahme ¬(2)

Damit ist für $1\leq l\leq m$ $\alpha[j_l]E_a\alpha[i]$ und insbesondere $\alpha[j]=\alpha[j_m]E_a\alpha[i]$ gezeigt. $\alpha[j]E_a\alpha[i]$ steht jedoch im Widerspruch zur Annahme, denn in diesem Fall existiert ein $\alpha[k]\in Z_a(\bigcup_{\alpha[i]\in W(\alpha)\&f_a(\alpha[i])} F_{w(\alpha)}(\alpha[i]))$ mit der Eigenschaft:

$$t(\alpha[k])\geq t(\alpha[i]) \;\&\; \alpha[k]E_a\alpha[j],$$

nämlich $\alpha[i]$ selbst. ■

Satz 5.4.2/1: Sei αx, $x\in A$, eine beliebige aktive Berechnungsfolge des aus n Prozessen bestehenden Prozeßsystems $P=(A,D,w)$. Für alle Aktionen $\alpha[i],\alpha[j]\in W(\alpha)$ gelte:

$$(*)\ (\alpha[j]B_a\alpha[i] \;\&\; \neg\alpha[j]E_a\alpha[i]) \Rightarrow (\forall\alpha[k]\in\{\alpha[l] \mid \alpha[l]\in W(\alpha) \;\&\; \alpha[l]E_a\alpha[i]\}\colon [t(\alpha[k])<t(\alpha[j])]).$$

Des weiteren existiere eine *maximale Recoveryeinheitsgröße* $g_{re}\in\mathbb{N}$, so daß für alle aktiven Berechnungen α von P gilt:

$$g_{re_\alpha} := \max_{\alpha[k]\in W(\alpha)} |\{\alpha[j] \mid \alpha[j]\in W(\alpha) \;\&\; \alpha[j]E_a\alpha[k]\}| < g_{re}.$$

Wird im Zustand $val_P(\alpha x)$ die Zurücksetzung des Prozesses $P_{bs(x)}$ zum Wiederaufsetzpunkt $WP_{\alpha x}(x)$ erforderlich, so kann bei Nichtberücksichtigung potentieller Fehlerverursacher die Zurücksetzung einer Aktion $\alpha[i]\in W(\alpha)$ die Zurücksetzung von maximal m Aktionen $\alpha[j_1],\ldots,\alpha[j_m]$, $m<g_{re}-1$, mit der Eigenschaft $t(\alpha[j_l])<t(\alpha[i])$, $1\leq l\leq m$, eines jeden Prozesses P_k von P erfordern.

Beweis: Angenommen dies wäre nicht der Fall, d.h. die Zurücksetzung von $\alpha[i]$ würde die Zurücksetzung von $m\geq g_{re}-1$ Aktionen $\alpha[j_1],\ldots,\alpha[j_m]$ mit der Eigenschaft $t(\alpha[j_l])<t(\alpha[i])$ eines Prozesses P_i erforderlich machen.
Nach Hilfssatz 5.4.2/1-(2) muß zu jedem $\alpha[j_l]$ ein $\alpha[k_l]$ existieren mit der Eigenschaft:

$$\alpha[j_l]E_a\alpha[k_l] \;\&\; t(\alpha[k_l])\geq t(\alpha[i]).$$

Sei $q\in\{1,\ldots,m\}$ so gewählt, daß $t(\alpha[j_q])=min\{t(\alpha[j_l]) \mid 1\leq l\leq m\}$.
Da $\alpha[j_q]E_a\alpha[k_q]$, d.h. $WP_a(\alpha[j_q])=WP_a(\alpha[k_q])$ gilt, die $\alpha[j_l]$, $1\leq j\leq m$, alle zum gleichen Prozeß gehören und $t(\alpha[j_l])\in[t(\alpha[j_q]),t(\alpha[k_q])[$ gilt, folgt, daß $WP_a(\alpha[j_l])=WP_a(\alpha[k_q])$

gilt und somit $m+1 \geq g_{re}$ Aktionen $\alpha[j_1],\ldots,\alpha[j_m],\alpha[k_q]$ zur gleichen Recoveryeinheit gehören. Dies steht im Widerspruch zur Annahme, die maximale Recoveryeinheitsgröße von α sei $g_{re_\alpha} < g_{re}$. ■

Korollar 5.4.2/1: Sei αx, $x \in A$, eine beliebige aktive Berechnungsfolge des Prozeßsystems $P=(A,D,w)$. Für alle Aktionen $\alpha[i],\alpha[j] \in W(\alpha)$ gelte:

(*) $(\alpha[j]B_a\alpha[i]\ \&\ \neg\alpha[j]E_a\alpha[i]) \Rightarrow$
$(\forall\alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha)\ \&\ \alpha[l]E_a\alpha[i]\}: [t(\alpha[k]) < t(\alpha[j])])$.

Wird im Zustand $val_P(\alpha x)$ die Zurücksetzung des Prozesses $P_{bz(x)}$ zum Wiederaufsetzpunkt $WP_{ax}(x)$ erforderlich, so gilt bei Nichtberücksichtigung potentieller Fehlerverursacher:

$$\forall\alpha[j] \in Z_{ax}\Big(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nv(ax)}(\alpha[i])\Big): [xE_{ax}\alpha[j]].$$

Beweis: Folgt unmittelbar mit Hilfssatz 5.4.2/1-(2), denn nach diesem Hilfssatz muß gelten:

$$\forall\alpha[j] \in Z_{ax}\Big(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nv(ax)}(\alpha[i])\Big): [t(\alpha[j]) > t(x)]$$

oder es muß ein

$$\alpha[k] \in Z_{ax}\Big(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nv(ax)}(\alpha[i])\Big)$$

existieren mit der Eigenschaft:

$$\alpha[k]E_{ax}\alpha[j]\ \&\ t(\alpha[k]) \geq t(x).$$

Ein $\alpha[j] \in Z_{ax}\Big(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nv(ax)}(\alpha[i])\Big)$ mit der Eigenschaft $t(\alpha[j]) > t(x)$ kann es nach Voraussetzung (x ist die aktuellste Aktion!) nicht geben, und

$$\alpha[k] \in Z_{ax}\Big(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nv(ax)}(\alpha[i])\Big),\ k ::= |\alpha|+1,$$

ist die einzige Aktion, für die

$$\alpha[k]E_{ax}\alpha[j]\ \&\ t(\alpha[k]) \geq t(x)$$

gelten kann. ■

Korollar 5.4.2/2: Sei αx, $x \in A$, eine beliebige aktive Berechnungsfolge des Prozeßsystems $P=(A,D,w)$. Für alle Aktionen $\alpha[i],\alpha[j] \in W(\alpha)$ gelte:

$$(*)\ (\alpha[j]B_a\alpha[i]\ \&\ \neg\alpha[j]E_a\alpha[i])\ \Rightarrow$$
$$(\forall \alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha)\ \&\ \alpha[l]E_a\alpha[i]\}\colon [t(\alpha[k]) < t(\alpha[j])]).$$

Wird im Zustand $val_P(\alpha x)$ die Zurücksetzung des Prozesses $P_{bz(x)}$ zum Wiederaufsetzpunkt $WP_{ax}(x)$ erforderlich, so gilt bei Nichtberücksichtigung potentieller Fehlerverursacher:

$$\exists n \in \mathbb{N}\colon [|\{i \mid \alpha[i] \in W(\alpha)\ \&\ xE_{ax}\alpha[i]\}| < n\ \Rightarrow$$
$$|Z_{ax}(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nw(ax)}(\alpha[i]))| < n].$$

Beweis: Folgt unmittelbar mit Korollar 5.4.2/1 und der Voraussetzung:

$$|\{i \mid \alpha[i] \in W(\alpha)\ \&\ xE_{ax}\alpha[i]\}| < n.\ \blacksquare$$

Korollar 5.4.2/3: Sei αx, $x \in A$, eine beliebige aktive Berechnungsfolge des Prozeßsystems $P=(A,D,w)$. Für alle Aktionen $\alpha[i],\alpha[j] \in W(\alpha)$ gelte:

$$(*)\ (\alpha[j]B_a\alpha[i]\ \&\ \neg\alpha[j]E_a\alpha[i])\ \Rightarrow$$
$$(\forall \alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha)\ \&\ \alpha[l]E_a\alpha[i]\}\colon [t(\alpha[k]) < t(\alpha[j])]).$$

Wird im Zustand $val_P(\alpha x)$ die Zurücksetzung des Prozesses $P_{bz(x)}$ zum Wiederaufsetzpunkt $WP_{ax}(x)$ erforderlich, so gilt bei Nichtberücksichtigung potentieller Fehlerverursacher:

$$Z_{ax}(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nw(ax)}(\alpha[i])) = U_{Z_{ax}}(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nw(ax)}(\alpha[i])).$$

Beweis: Folgt unmittelbar mit Korollar 5.4.2/1:

$$\forall \alpha[j] \in Z_{ax}(\bigcup_{\alpha[i] \in W(ax)\,\&\, f_{ax}(\alpha[i])} F_{nw(ax)}(\alpha[i]))\colon [xE_{ax}\alpha[j]],$$

und Definition 4.2.1/1. ∎

Satz 5.4.2/2: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen, und für jede aktive Berechnungsfolge α von P gelte:

$$(*)\ \forall \alpha[i],\alpha[j] \in W(\alpha)\colon [(\alpha[j]B_a\alpha[i]\ \&\ \neg\alpha[j]E_a\alpha[i])\ \Rightarrow$$
$$(\forall$$
$$\alpha[k] \in \{\alpha[l] \mid \alpha[l] \in W(\alpha)\ \&\ \alpha[l]E_a\alpha[i]\}\colon [t(\alpha[k]) < t(\alpha[j])])].$$

Des weiteren gebe es Konstanten $n_1, n_2 \in \mathbb{N}$, so daß für jede aktive Berechnung α von P gelte:

(1) $\forall \alpha[j] \in W(\alpha)$: $|\{i \mid \alpha[i] \in W(\alpha) \;\&\; \alpha[j] E_\alpha \alpha[i]\}| < n_1$ und

(2) in bezug auf alle als fehlerhaft erkannten Aktionen $\alpha[j] \in W(\alpha)$:

$$|Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[j]), t(\alpha[|\alpha|])])| < n_2.$$

Unter diesen Voraussetzungen ist P dominoeffektfrei, d.h. für jede beliebige aktive Berechnung α von P gilt:

$$|Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))| < n^* (n_1 - 1) + n_2.$$

Beweis: Die Größe von $|Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))|$ läßt sich wie folgt abschätzen:

$$|Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]))| \leq |Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[1]), t(\alpha[j])[])| +$$
$$|Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[j]), t(\alpha[|\alpha|])])| <$$
$$|Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[1]), t(\alpha[j])[])| + n_2.$$

Bleibt zu zeigen:

$$|Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{nv(\alpha)}(\alpha[i]) \mid [t(\alpha[1]), t(\alpha[j])[])| < n^* (n_1 - 1).$$

Letzteres folgt unmittelbar mit Korollar 5.4.2/4. ■

Korollar 5.4.2/4: Unter den Voraussetzungen des vorangehenden Satzes 5.4.2/1 kann die Zurücksetzung einer Aktion $\alpha[i] \in W(\alpha)$ lediglich die Zurücksetzung von weniger als $n^* (g_{TW} - 1)$ Aktionen $\alpha[j_l]$ mit der Eigenschaft $t(\alpha[j_l]) < t(\alpha[i])$ erfordern.

Beweis: Folgt unmittelbar mit Satz 5.4.2/1 und der Annahme das Prozeßsystem P bestehe aus n Prozessen. ■

5.5. Rein azyklische Abhängigkeiten

Die bisherigen Überlegungen in den Abschnitten 5.1 bis 5.4 haben gezeigt, daß die Beschränkung der zyklischen Zurücksetzung bzw. das Nichtauftreten des Dominoeffektes durch geeignete Anordnung von Wiederaufsetzpunkten während einer Berechnung erreicht werden können.
Im folgenden wird nun am Beispiel der Eigenschaften rein azyklischer Prozeßbeziehungen dargelegt, daß Wiederaufsetzpunkte nicht das einzige Mittel gegen die Gefahr der unbeschränkbaren zyklischen Zurücksetzung und des Dominoeffektes darstellen, sondern gelegentlich auch einfachere Möglichkeiten zum Ausschluß der vorgenannten Gefahren bestehen.
Zu diesem Zweck werden zunächst im Abschnitt 5.5.1 verschiedene, rein azyklische Prozeßbeziehungen kennzeichnende Merkmale abgeleitet, bei deren Vorhandensein zyklische Zurücksetzung nicht auftreten kann, und im Abschnitt 5.5.2 wird anschließend ihre Bedeutung hinsichtlich des Ausschlusses der Dominoeffektgefahr skizziert.

Die Idee zur Vermeidung/Verhinderung des Dominoeffektes mittels rein azyklischer Prozeßbeziehungen wird auch in [Rus80] am Beispiel von Erzeuger-/Verbrauchersystemen diskutiert (vgl. Kapitel 6).

5.5.1. Charakterisierung rein azyklischer Abhängigkeiten

Im folgenden wird im Rahmen der Hilfssätze 5.5.1/1 bis 5 zu fünf unterschiedlichen, rein azyklische Prozeßbeziehungen kennzeichnenden Merkmalen (Bedingungen) der Nachweis des Nichtauftretens der zyklischen Zurücksetzung im Falle ihres Vorhandenseins (ihrer Einhaltung) geführt. Diese Zwischenergebnisse ermöglichen schließlich den Ausschluß der Dominoeffektgefahr für Prozeßsysteme, deren Berechnungen die skizzierten Merkmale aufweisen, im nächsten Abschnitt.

Hilfssatz 5.5.1/1: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1,\ldots,P_n$. Gilt für eine aktive Berechnungsfolge α von P:

$$(*)\ \forall i,j\in\{1,\ldots,|\alpha|\}\colon [\alpha[j]B_\alpha\alpha[i]\ \Rightarrow\ bz(\alpha[i])\leq bz(\alpha[j])],$$

dann kann von

$$(**)\ \exists\alpha[l_1],\ldots,\alpha[l_m]\in Z_\alpha\Big(\bigcup_{\alpha[i]\in V(\alpha)\,\&\,f_\alpha(\alpha[i])} F_{w(\alpha)}(\alpha[i])\Big)\colon$$
$$[\alpha[l]=\alpha[l_m](E_\alpha\cup B_\alpha)\alpha[l_{m-1}]\ldots\alpha[l_2](E_\alpha\cup B_\alpha)\alpha[l_1]=\alpha[k]]$$

auf

$$\forall i \in \{1,\ldots,n\}: [|\{ WP_\alpha(\alpha[l_j]) \mid \exists \alpha[l_k] \in \{\alpha[l_1],\ldots,\alpha[l_{j-1}]\} \cap P_i : \alpha[l_j] \in \{\alpha[l_1],\ldots,\alpha[l_m]\} \cap P_i \;\&\; t(WP_\alpha(\alpha[l_j])) < t(WP_\alpha(\alpha[l_k]))\}| = 0]$$

geschlossen werden.

Beweis: Da, nach Definition der Relation E_α, von $\alpha[l_\tau] E_\alpha \alpha[l_{\tau-1}]$ auf $bz(\alpha[l_\tau]) = bz(\alpha[l_{\tau-1}])$ $\tau \in \{1, \ldots, m\}$, geschlossen werden kann, und nach Voraussetzung (*) von $\alpha[l_\tau] B_\alpha \alpha[l_{\tau-1}]$ der Schluß auf $bz(\alpha[l_\tau]) \geq bz(\alpha[l_{\tau-1}])$ zulässig ist, folgt mit der Voraussetzung (**)

$$(***) \quad bz(\alpha[l]) = bz(\alpha[l_m]) \geq bz(\alpha[l_{m-1}]) \geq \ldots \geq bz(\alpha[l_2]) \geq bz(\alpha[l_1]) = bz(\alpha[k]).$$

Zum Beweis der Aussage des Hilfssatzes genügt es zu zeigen, daß es kein Aktionenpaar $\alpha[l]$, $\alpha[k] \in \{\alpha[l_1],\ldots,\alpha[l_m]\}$ geben kann, für das gilt:

$$proc_P(\alpha[l_j], \alpha[l_k]) \;\&\; k < j \;\&\; t(WP_\alpha(\alpha[l_k])) > t(WP_\alpha(\alpha[l_j])).$$

Angenommen es gäbe Aktionen $\alpha[l_j]$ und $\alpha[l_k]$ mit der beschriebenen Eigenschaft, dann muß, da nach Definition von $proc_P(\alpha[l_j], \alpha[l_k])$ in diesem Falle $bz(\alpha[l_j]) = bz(\alpha[l_k])$ gilt, wegen (***) gelten:

$$\tau \in \{k+1,\ldots j\} \Rightarrow proc_P(\alpha[l_\tau], \alpha[l_{\tau-1}]).$$

Bleibt zu zeigen, daß für $\tau \in \{k+1, \ldots, j\}$ $t(WP_\alpha(\alpha[l_\tau])) \geq t(WP_\alpha(\alpha[l_{\tau-1}]))$ gelten muß, denn in diesem Falle läßt sich mit Hilfe der vollständigen Induktion auf

$$t(WP_\alpha(\alpha[l_j])) \geq t(WP_\alpha(\alpha[l_k]))$$

schließen, im Widerspruch zur Annahme

$$t(WP_\alpha(\alpha[l_k])) > t(WP_\alpha(\alpha[l_j])).$$

Da $\alpha[l_\tau](E_\alpha \cup B_\alpha)\alpha[l_{\tau-1}]$ für $\tau \in \{k+1, \ldots, j\}$ gilt, läßt sich $t(WP_\alpha(\alpha[l_\tau])) \geq t(WP_\alpha(\alpha[l_{\tau-1}]))$ durch Unterscheidung zweier Fälle wie folgt zeigen:

Fall 1 $(\alpha[l_\tau] B_\alpha \alpha[l_{\tau-1}])$. Aufgrund der Definition von B_α folgt $t(\alpha[l_\tau]) > t(\alpha[l_{\tau-1}])$ und, da für $\tau \in \{k+1,\ldots,j\}$ $proc_P(\alpha[l_\tau], \alpha[l_{\tau-1}])$ gilt, folgt insgesamt:

$$t(WP_\alpha(\alpha[l_\tau])) \geq t(WP_\alpha(\alpha[l_{\tau-1}])).$$

Fall 2 $(\alpha[l_\tau] E_\alpha \alpha[l_{\tau-1}])$ Aufgrund der Definition von E_α kann von $\alpha[l_\tau] E_\alpha \alpha[l_{\tau-1}]$ auf $t(WP_\alpha(\alpha[l_\tau])) = t(WP_\alpha(\alpha[l_{\tau-1}]))$ geschlossen werden. ■

Hilfssatz 5.5.1/2: Sei $P = (A, D, w)$ ein Prozeßsystem von n Prozessen $P_1, \ldots, P_n$. Gilt für eine aktive Berechnungsfolge α von P:

$$(*)\ \forall i,j,k,l\in\{1,\ldots,|\alpha|\}:\ [(\alpha[j]B_a\alpha[i]\ \&\ \alpha[l]B_a\alpha[k])\ \Rightarrow\ (\neg proc_P(\alpha[i],\alpha[l])\ \|\ proc_P(\alpha[i],\alpha[j]))],$$

dann kann von

$$(**)\ \exists\alpha[l_1],\ldots,\alpha[l_m]\in Z_a(\bigcup_{\alpha[i]\in W(a)\&f_a(\alpha[i])} F_{nv(a)}(\alpha[i])):\ [\alpha[l]=\alpha[l_m](E_a\cup B_a)\alpha[l_{m-1}]\ldots\alpha[l_2](E_a\cup B_a)\alpha[l_1]=\alpha[k]]$$

auf

$$\forall i\in\{1,\ldots,n\}:\ [|\{WP_a(\alpha[l_j])\mid \exists\alpha[l_k]\in\{\alpha[l_1],\ldots,\alpha[l_{j-1}]\}\cap P_i.\ \alpha[l_j]\in\{\alpha[l_1],\ldots,\alpha[l_m]\}\cap P_i\ \&\ t(WP_a(\alpha[l_j]))<t(WP_a(\alpha[l_k]))\}|=0]$$

geschlossen werden

Beweis: Zum Beweis der Aussage des Hilfssatzes genügt es zu zeigen, daß es kein Aktionenpaar $\alpha[l_j],\alpha[l_k]\in\{\alpha[l_1],\ldots,\alpha[l_m]\}$ geben kann, für das gilt:

$$proc_P(\alpha[l_j],\alpha[l_k])\ \&\ k<j\ \&\ t(WP_a(\alpha[l_k]))>t(WP_a(\alpha[l_j])).$$

Angenommen es gäbe Aktionen $\alpha[l_j]$ und $\alpha[l_k]$ mit der beschriebenen Eigenschaft, dann muß $k+1\neq j$ gelten, da sonst $\alpha[l_j](E_a\cup B_a)\alpha[l_k]$ erfüllt wäre, und aufgrund von $proc_P(\alpha[l_j],\alpha[l_k])$ analog der Fallunterscheidung am Ende des Hilfssatzes 5.5.1/1 auf

$$t(WP_a(\alpha[l_j]))\geq t(WP_a(\alpha[l_k])),$$

im Widerspruch zur Annahme

$$t(WP_a(\alpha[l_j]))<t(WP_a(\alpha[l_k])),$$

geschlossen werden könnte.
Da des weiteren $\alpha[l_{k+1}](E_a\cup B_a)\alpha[l_k]\ \&\ \alpha[l_j](E_a\cup B_a)\alpha[l_{j-1}]$ nur gelten kann, falls $proc_P(\alpha[l_k],\alpha[l_{k+1}])$ oder $proc_P(\alpha[l_j],\alpha[l_{j-1}])$ erfüllt ist (andernfalls würde aufgrund der Voraussetzung (*) $\neg proc_P(\alpha[l_j],\alpha[l_k])$ folgen, was einen Widerspruch zur Annahme darstellen würde), kann auf:

$$t(WP_a(\alpha[l_k]))\leq t(WP_a(\alpha[l_{k+1}]))$$

oder

$$t(WP_a(\alpha[l_j]))\geq t(WP_a(\alpha[l_{j-1}]))$$

geschlossen werden. Das heißt eines der Aktionenpaare $\alpha[l_{j-1}],\alpha[l_k]$ oder $\alpha[l_j],\alpha[l_{k+1}]$ muß ebenfalls (mit der Umbenennung $j::=j-1$ oder $k::=k+1$) die Ungleichung $k+1\neq j$ erfüllen.
Da sowohl $(j-1)-k<j-k$ als auch $j-(k+1)<j-k$ gilt, läßt sich letzte Folgerung entweder mit der Umbenennung $j::=j-1$ oder mit der Umbenennung $k::=k+1$ unter

Nutzung des Ergebnisses $k+1 \neq j$ nach endlich vielen Schritten auf einen Widerspruch zur Annahme führen. ■

Hilfssatz 5.5.1/3: Sei $P=(A,D,w)$ ein Prozeßsystem von n Prozessen $P_1, \ldots, P_n$. Gilt für eine aktive Berechnungsfolge α von P:

$$(*) \quad \neg \exists i,j \in \{1,\ldots,|\alpha|\}: [\alpha[i](E_\alpha \cup B_\alpha)^* \alpha[j](E_\alpha \cup B_\alpha)^* \alpha[i] \ \& \ \neg proc_P(\alpha[i],\alpha[j])]$$

dann kann von

$$(**) \quad \exists \alpha[l_1],\ldots,\alpha[l_m] \in Z_\alpha(\bigcup_{\alpha[i] \in W(\alpha) \& f_\alpha(\alpha[i])} F_{rw(\alpha)}(\alpha[i])): \\ [\alpha[l_m](E_\alpha \cup B_\alpha)\alpha[l_{m-1}] \ldots \alpha[l_2](E_\alpha \cup B_\alpha)\alpha[l_1]]$$

auf

$$\forall i \in \{1,\ldots,n\}: [|\{WP_\alpha(\alpha[l_j]) \mid \exists \alpha[l_k] \in \{\alpha[l_1],\ldots,\alpha[l_{j-1}]\} \cap P_i: \\ \alpha[l_j] \in \{\alpha[l_1],\ldots,\alpha[l_m]\} \cap P_i \ \& \ t(WP_\alpha(\alpha[l_j])) < t(WP_\alpha(\alpha[l_k]))\}| = 0]$$

geschlossen werden.

Beweis: Angenommen dieser Schluß wäre unzulässig. Dann muß es mindestens zwei Aktionen $\alpha[l_k],\alpha[l_q] \in \{\alpha[l_1],\ldots,\alpha[l_m]\} \cap P_k$ geben, mit $k \in \{1,\ldots,n\}$ und $i<q$, für die $t(WP_\alpha(\alpha[l_k])) > t(WP_\alpha(\alpha[l_q]))$ und somit auch $WP_\alpha(\alpha[l_k]) \neq WP_\alpha(\alpha[l_q])$ gilt.
Von $t(WP_\alpha(\alpha[l_k])) > t(WP_\alpha(\alpha[l_q])) \ \& \ proc_P(\alpha[l_k],\alpha[l_q])$ kann auf $t(\alpha[l_k]) > t(\alpha[l_q])$ geschlossen werden, und, da $\alpha[l_q](E_\alpha \cup B_\alpha)^* \alpha[l_k]$, aber nicht $\alpha[l_q] E_\alpha \alpha[l_k]$ gilt, muß eine Aktion $\alpha[l_j]$, $i<j<q$, existieren, so daß $\neg proc_P(\alpha[l_k],\alpha[l_j])$ und

$$\alpha[l_q](E_\alpha \cup B_\alpha)^* \alpha[l_j](E_\alpha \cup B_\alpha)^* \alpha[l_k]$$

gilt. Des weiteren stehen aufgrund von $t(\alpha[l_k]) > t(\alpha[l_q]) \ \& \ proc_P(\alpha[l_k],\alpha[l_q])$ die Aktionen $\alpha[l_k]$ und $\alpha[l_q]$ auch in der Beziehung $\alpha[l_k](E_\alpha \cup B_\alpha)^* \alpha[l_q]$.
Insgesamt bedeutet dies (mit $i ::= l_i \ \& \ j ::= l_j$):

$$\exists i,j \in \{1,\ldots,|\alpha|\}: [\alpha[i](E_\alpha \cup B_\alpha)^* \alpha[j](E_\alpha \cup B_\alpha)^* \alpha[i] \ \& \ \neg proc_P(\alpha[i],\alpha[j])]$$

im Widerspruch zur Voraussetzung (*). ■

Hilfssatz 5.5.1/4: Ersetzt man in Hilfssatz 5.5.1/3 die Voraussetzung (*) durch:

$$\neg \exists i,j \in \{1,\ldots,|\alpha|\}: [\alpha[j](E_\alpha \cup B_\alpha)^* \alpha[i] \ \& \ proc_P(\alpha[i],\alpha[j]) \ \& \ t(\alpha[i]) > t(\alpha[j]) \ \& \\ WP_\alpha(\alpha[i]) \neq WP_\alpha(\alpha[j])],$$

so bleibt die Aussage des Hilfssatzes 5.5.1/3 unverändert gültig.

Beweis: Angenommen die Aussage des Hilfssatzes 5.5.1/3 würde bei Ersetzung der Voraussetzung (*) durch obige Bedingung ihre Gültigkeit verlieren. Dann müßte es mindestens zwei Aktionen $\alpha[l_i],\alpha[l_j]\in\{\alpha[l_1],\ldots,\alpha[l_m]\}\cap P_k$ geben, mit $k\in\{1,\ldots,n\}$ und $i<j$, für die $t(WP_a(\alpha[l_i]))>t(WP_a(\alpha[l_j]))$ und somit auch $WP_a(\alpha[l_i])\neq WP_a(\alpha[l_j])$ gelten würde.
Von $t(WP_a(\alpha[l_i]))>t(WP_a(\alpha[l_j]))$ & $proc_P(\alpha[l_i],\alpha[l_j])$ könnte dann weiter auf $t(\alpha[l_i])>t(\alpha[l_j])$ geschlossen werden.
Insgesamt würde dies bedeuten (mit $i::=l_i$ & $j::=l_j$):

$$\exists i,j\in\{1,\ldots,|\alpha|\}:\ [\alpha[j](E_a\cup B_a)^*\alpha[i]\ \&\ proc_P(\alpha[i],\alpha[j])\ \&\ t(\alpha[i])>t(\alpha[j])\ \&\ WP_a(\alpha[i])\neq WP_a(\alpha[j])]$$

im Widerspruch zur obigen Bedingung. ■

Hilfssatz 5.5.1/5: Sei α eine aktive Berechnungsfolge eines Prozeßsystems $P=(A,D,w)$. Stellt die Relation $(E_a\cup B_a)^*$ eingeschränkt auf die Äquivalenzklassen der Relation E_a eine *partielle Ordnungsrelation* dar, so kann von:

$$\exists\alpha[l_1],\ \ldots,\alpha[l_m]\in Z_a\Big(\bigcup_{\alpha[i]\in W(a)\,\&\,f_a(\alpha[i])} F_{w(a)}(\alpha[i])\Big):\ [\alpha[l]=\alpha[l_m](E_a\cup B_a)\alpha[l_{m-1}]\ldots\alpha[l_2](E_a\cup B_a)\alpha[l_1]=\alpha[k]]$$

auf

$$\forall i\in\{1,\ldots,n\}\ [|\{WP_a(\alpha[l_j])\mid\exists\alpha[l_k]\in\{\alpha[l_1],\ldots,\alpha[l_{j-1}]\}\cap P_i:\ \alpha[l_j]\in\{\alpha[l_1],\ldots,\alpha[l_m]\}\cap P_i\ \&\ t(WP_a(\alpha[l_j]))<t(WP_a(\alpha[l_k]))\}|=0]$$

geschlossen werden.

Beweis: Es genügt zu zeigen, daß nicht gelten kann:

$$(*)\ \exists i,j\in\{1,\ldots,|\alpha|\}:\ [\alpha[i](E_a\cup B_a)^*\alpha[j](E_a\cup B_a)^*\alpha[i]\ \&\ \neg proc_P(\alpha[i],\alpha[j])].$$

Die Aussage des Hilfssatzes ergibt sich dann als Folgerung von Hilfssatz 5.5.1/3.
Würde (*) gelten, so könnte, da die Relation $(E_a\cup B_a)^*$ eingeschränkt auf die Relation E_a nach Voraussetzung partielle Ordnungsrelation und daher insbesondere antisymmetrisch ist, von $\alpha[i](E_a\cup B_a)^*\alpha[j]$ & $\alpha[j](E_a\cup B_a)^*\alpha[i]$ auf $\alpha[i]E_a\alpha[j]$ ($\alpha[i]$ und $\alpha[j]$ gehören zur gleichen Äquivalenzklasse) geschlossen werden im Widerspruch zu $\neg proc_P(\alpha[i],\alpha[j])$. ■

Erläuterung: Durch die Einschränkung der Relation $(E_a\cup B_a)^*$ auf die Äquivalenzklassen der Äquivalenzrelation E_a soll zum Ausdruck gebracht werden, daß alle Aktionen einer Recoveryeinheit in Hilfssatz 5.5.1/5 als *identisch* zu betrachten sind. Konkret bedeutet dies, daß man von $\alpha[i](E_a\cup B_a)^*\alpha[j]$ & $\alpha[j](E_a\cup B_a)^*\alpha[i]$ auf $\alpha[i]E_a\alpha[j]$ (Antisymmetrie-Eigenschaft) schließen kann.

5.5.2. Wirksamkeit rein azyklischer Abhängigkeiten

Eine Situation, in welcher der Dominoeffekt im Falle des Vorhandenseins rein azyklischer Prozeßbeziehungen, wie sie in Abschnitt 5.5.1 charakterisiert werden, nicht auftreten kann, beschreibt schließlich der folgende Satz, mit dem die Folge der unter Nichtberücksichtigung potentieller Fehlerverursacher abgeleiteten Beispiele dominoeffektfreier Prozeßsysteme ohne Anspruch auf Vollständigkeit abgeschlossen wird.

Satz 5.5.2/1: Sei αx, $x \in A$, eine beliebige aktive Berechnungsfolge des aus n_2 Prozessen bestehenden Prozeßsystems $P=(A,D,w)$. Wird im Zustand $val_P(\alpha x)$ die Zurücksetzung des Prozesses $P_{bz(x)}$ zum Wiederaufsetzpunkt $WP_{\alpha x}(x)$ erforderlich, so kann bei Nichtberücksichtigung potentieller Fehlerverursacher unter den weiter unten näher bezeichneten Voraussetzungen (1) und (2) der Dominoeffekt nicht auftreten, d.h.:

$$(1)\ \&\ (2)\ \Rightarrow\ |Z_{\alpha x}(\bigcup_{\alpha[i]\in W(\alpha x)\&f_{\alpha x}(\alpha[i])} F_{nv(\alpha x)}(\alpha[i]))| \leq \prod_{j=1}^{2} n_j,$$

wobei $n_i \in \mathbf{N}$ eine prozeßsystemspezifische, von der Länge der Berechnungsfolge α unabhängige Konstante und (1), (2) wie folgt definiert sind:

(1) $\exists n_1 \in \mathbf{N} \forall \beta \forall \beta[j] \in W(\beta)$: $[(\beta$ aktiv in $P) \Rightarrow$

$$(|Z_\beta(\bigcup_{\beta[i]\in W(\beta)\&f_\beta(\beta[i])} F_{nv(\beta)}(\beta[i]) \mid [t(WP_\beta(\beta[j])), t(\beta[j])])| < n_1)],$$

(2) $\forall i,j \in \{1,\ldots,|\alpha|\}$: $[\alpha[j]B_a\alpha[i] \Rightarrow bz(\alpha[i]) \leq bz(\alpha[j])]$ (vgl. (*) in Hilfssatz 5.5.1/1)

(Eine Alternative zu (2) wäre auch die Forderung (vgl. (*) in Hilfssatz 5.5.1/2):

$\forall i,j,k,l \in \{1,\ldots,|\alpha|\}$: $[\alpha[j]B_a\alpha[i]\ \&\ \alpha[l]B_a\alpha[k] \Rightarrow (\neg proc_P(\alpha[i],\alpha[l]) \parallel proc_P(\alpha[i],\alpha[j]))]$

oder eine der entsprechenden Voraussetzungen der Hilfssätze 5.5.1/3, 4 bzw. 5).

Beweis: Unter den gemachten Voraussetzungen, α endliche aktive Berechnungsfolge, Zurücksetzung des Prozesses $P_{bz(x)}$ zum Wiederaufsetzpunkt $WP_{\alpha x}(x)$ und Punkt (2), folgt mit Hilfe der Hilfssätze 5.5.1/1, 2, 3, 4 bzw. 5:

$$(*)\ \forall j \in \{1,\ldots,n_2\} \forall \alpha x[l] \in Z_{\alpha x}(\bigcup_{\alpha[i]\in W(\alpha x)\&f_{\alpha x}(\alpha[i])} F_{nv(\alpha x)}(\alpha[i])) \cap P_j:$$
$$[\exists \alpha[l_1],\ldots,\alpha[l_m]: [\alpha x[l]=\alpha[l_m](E_a \cup B_a)\alpha[l_{m-1}]\ldots\alpha[l_2](E_a \cup B_a)\alpha[l_1]=x \Rightarrow$$
$$(|\{WP_a(\alpha[l_q]) \mid \exists \alpha[l_k] \in \{\alpha[l_1],\ldots,\alpha[l_{q-1}]\} \cap P_j:$$
$$[\alpha[l_q] \in \{\alpha[l_1],\ldots,\alpha[l_m]\} \cap P_j\ \&\ t(WP_a(\alpha[l_q])) < t(WP_a(\alpha[l_k]))]\}| = 0)]]$$

Aufgrund von (1) und (*) folgt die Aussage des Satzes schließlich mit Satz 4.2.3/2. ■

5.6. Berücksichtigung potentieller Fehlerverursacher

Es bleibt dem Leser überlassen, zu den Sätzen des Kapitels fünf inhaltlich vergleichbare Aussagen auch für den Fall der Berücksichtigung potentieller Fehlerverursacher abzuleiten. Entsprechende Voraussetzungen dafür wurden im Abschnitt 4.3 geschaffen.

Im Rahmen dieser Arbeit wird auf eine Vertiefung des Punktes Berücksichtigung potentieller Fehlerverursacher über das in Kapitel 4 Gesagte hinaus verzichtet, da eine detailliertere Beschreibung gegen eine Fehlerausbreitung zu treffender Vorkehrungen, welche das zentrale Problem in diesem Zusammenhang darstellt, anhand von Beispielprozeßsystemen unter Einsatz ähnlicher Überlegungen, wie der in diesem Kapitel angewandten, problemlos möglich ist. So kann z.B. die Fehlerausbreitung durch sorgfältiges Auswahlen und Anordnen von Akzeptanztests, die in diesem Sinne als dual zu Wiederaufsetzpunkten bei der Rücksetzausbreitung zu sehen sind, innerhalb von Berechnungen eingegrenzt werden. Das heißt eine Untersuchung würde keine weiteren grundlegend neuen Erkenntnisse mit sich bringen. Zudem würde sie den Umfang dieser Abhandlung bei weitem sprengen.
Aus diesen Gründen erscheint es zweckmäßiger dem Leser stattdessen zum Abschluß einen Eindruck davon zu vermitteln, welche Ergebnisse zum Phänomen Dominoeffekt bisher in Veröffentlichungen anderer Autoren vorgestellt wurden, und wie sich der bisher erreichte Kenntnisstand darstellt.
Insbesondere auf den letztgenannten Aspekt geht das nachfolgende vorletzte Kapitel dieser Arbeit ein, welches den wesentlichen Inhalt von Forschungsberichten, Dissertationen und Aufsätzen von drei Verfassern, die den aktuellen Stand der Forschung auf dem diskutierten Gebiet entscheidend geprägt haben, wiedergibt. Eine Zusammenfassung mit Schlußbemerkungen im siebten und letzten Kapitel rundet die vorliegende Ausarbeitung schließlich ab.

Auf den folgenden Spezialfall sei hier jedoch noch hingewiesen: Bei sofortiger Verifizierung von gelesener globaler Information, der Verifizierung lokaler Information jeweils vor Abschluß einer jeden Recoveryeinheit und korrekter Arbeitsweise von Akzeptanztests, kann der Fall der Berücksichtigung potentieller Fehlerverursacher auf denjenigen der Nichberücksichtigung potentieller Fehlerverursacher zurückgeführt werden.

6. Ergebnisse themaverwandter Arbeiten

In den folgenden Abschnitten dieses Kapitels werden Bezüge zu im Vergleich zu den vorliegenden Ausführungen ähnlich gearteten Arbeiten dreier Autoren hergestellt. Hierbei ist es das Ziel zu zeigen, daß die wesentlichen Ergebnisse verwandter Beiträge durch die in dieser Arbeit bewiesenen Sätze abgedeckt werden und nur einen Teilaspekt der vorliegenden Dissertation ausmachen. Zu diesem Zweck werden die Arbeiten von K.H. Kim, [Kim78a], [Kim78b], [Kim78c], K. Kant, [Kan81], und D.L.Russell, [Rus76], [Rus77], [Rus80], untergliedert nach dem jeweils zugrundegelegten Prozeßsystem-/Fehlerbehandlungs-Modell und den wesentlichen Ergebnissen, in groben Zügen vorgestellt. Ergänzende Anmerkungen vervollständigen diese Angaben.

Als erstes wird auf die Beiträge von K.H. Kim zur Konstruktion eines benutzertransparenten Schemas zur Prozeßablauf-Sicherung näher eingegangen. Anschließend folgen kurze Inhaltsangaben zu den Arbeiten von K. Kant und von D.L. Russell.

6.1. Kim's Prozeßablauf-Sicherungs-Schema

K.H. Kim geht es bei seinen Überlegungen in [Kim78a], [Kim78b], [Kim78c] darum, ein Schema zur benutzertransparenten Prozeßablauf-Sicherung zu finden (mit Benutzer ist hier auch der Anwendungs- und System-Programmierer gemeint), durch das der Dominoeffekt vermieden werden kann. Dazu bedient er sich der folgenden Modellbildung.

6.1.1. Kim's Prozeßsystem-Modell

In [Kim78a], [Kim78b], [Kim78c] legt K.H. Kim seinen Überlegungen das folgende Prozeßsystem-Modell zugrunde: Jeder Prozeß ist aus einzelnen Recoveryblöcken, vgl. [Hor74], [HR73], [Ran75] und [Ran77], aufgebaut; Prozesse können nur über Monitore, vgl. [Han73], [Hoa74], kommunizieren; Prozessen gemeinsame Betriebsmittel werden ausschließlich über Monitore verwaltet.

6.1.2. Kim's Fehlerbehandlungs-Modell

Um im Fehlerfall eine Fehlerkorrektur vornehmen zu können, wird vor Ausführung eines jeden Recoveryblockes zum ausführenden Prozeß ein prozeßlokaler

Wiederaufsetzpunkt, *base Recovery Point* (kurz: *base RP*) genannt, angelegt. Unter bestimmten Bedingungen werden zusätzlich zu den *base RPs* auch prozeßlokale Wiederaufsetzpunkte unmittelbar vor den lesenden Monitorbezügen der Prozesse eingerichtet. Zur Unterscheidung von den *base RPs* werden derartige Wiederaufsetzpunkte als *branch RPs* bezeichnet.
Die Entscheidung, wann ein *branch RP* einzurichten ist, wird wie folgt getroffen: Während der Ausführung eines Recoveryblockes $RB_{i,j}$ des Prozesses P_i werden alle diejenigen Recoveryblöcke registriert, von deren erfolgreichem Abschluß die Korrektheit der Ausführung von $RB_{i,j}$ abhängt. Die Menge dieser Recoveryblöcke wird als *Potential Recaller Set* von $RB_{i,j}$ (kurz: $PR^*(RB_{i,j})$) bezeichnet.
Ein Recoveryblock $RB_{i,j}$ ist von einem anderen Recoveryblock $RB_{k,l}$ genau dann abhängig, wenn $RB_{i,j}$ Daten über einen Monitor erhält, zu deren Entzug eine Rücksetzung von $RB_{k,l}$ führen würde.
Bei einem Monitorprozeduraufruf, über den ein Prozeß P_i im Recoveryblock $RB_{i,j}$ von einem Monitor verwaltete Daten liest, können neue Abhängigkeiten von $RB_{i,j}$ zu Recoveryblocken anderer Prozesse entstehen, d.h. die Menge $PR^*(RB_{i,j})$ kann wachsen. Ist dies der Fall, so wird unmittelbar vor diesem Monitoraufruf ein branch RP zu P_i angelegt.

Die Aufzeichnung eines Monitorzustandes zu einem Monitor M_j wird vor einem ändernden Monitoraufruf eines Prozesses P_i immer dann erforderlich, falls es sich um die erste von P_i initiierte Änderung von M_j seit dem Zeitpunkt der Erstellung des aktuellsten Wiederaufsetzpunktes zu P_i handelt. Die Aufzeichnung des Zustandes eines Monitors M_j erfolgt aber auch immer dann, wenn eine in einem Recoveryblock $RB_{i,j}$ des Prozesses P_i aktivierte Monitorprozedur von M_j einen lesenden Bezug zu Daten eines anderen Monitors M_k herstellt und sich durch diesen Vorgang die Menge $PR^*(RB_{i,j})$ vergrößert. In diesem Fall wird natürlich auch ein *branch RP* zum Prozeß P_i eingerichtet.

Über den korrekten Abschluß der Ausführung eines Recoveryblockes $RB_{i,j}$ entscheidet ein *Validierungstest* und der erfolgreiche Abschluß der zu $PR^*(RB_{i,j})$ gehörenden Recoveryblöcke.
Die Ausführung eines Recoveryblockes wird als *partiell verifiziert* (*"partially validated"*) bezeichnet, wenn der mit einem Recoveryblock $RB_{i,j}$ verbundene Akzeptanztest erfolgreich abgeschlossen wurde und $PR^*(RB_{i,j})$ nicht leer ist. Verläuft der Validierungstest positiv, und ist $PR^*(RB_{i,j})$ leer, so gilt $RB_{i,j}$ als *vollständig verifiziert* (*"completely validated"*), und $RB_{i,j}$ wird aus jeder Menge $PR^*(..)$, zu der er gehört, entfernt.

Im Fehlerfall unterscheidet Kim zwei Arten von Rücksetzausbreitung, die verschiedene Ursachen haben. Der erste Typ wird als *R-Propagation* bezeichnet und ist durch den Entzug von Daten, die bereits von anderen Prozessen weiterverarbeitet

wurden, bedingt (*"R-propagation is dued to the revocation of information sent out"*). Typ zwei wird als *S-Propagation* bezeichnet und beruht auf gegenseitigem Mißtrauen zwischen Prozessen (*"S-propagation is dued to suspicion"*); ein Prozeß kann einen anderen für sein Scheitern mitverantwortlich machen, falls er Information von diesem erhalten hat und ihm mißtraut.
Da, wie von Kim anhand eines Beispieles u.a. in [Kim78a] erläutert wird, S-Propagation im allgemeinen Fall ohne besondere Vereinbarungen nicht in den Griff zu bekommen ist, macht er die folgende Annahme: Die Korrektheit an andere Prozesse übergebener Information kann nur vom Sender selbst in Frage gestellt und anschließend entsprechend korrigiert werden; die Korrektheit validierter Daten in Frage zu stellen ist unzulässig. Durch diese Annahme schränkt Kim sein Augenmerk auf die Betrachtung der R-Propagation ein.

Wie die Bilder 6.1.2/1 und 2 zeigen, wird im Fehlerfall (in Bild 6.1.2/1 ist Prozeß P_3 als fehlerhaft angenommen), jeder fehlerhafte Prozeß zum aktuellsten *base RP* vor dem aufgetretenen Fehler zurückgesetzt. Danach werden alle Prozeßzustände und Monitorzustände (in Bild 6.1.2/1 gilt dies für die Prozesse P_1, P_2 und die Monitore M_1, M_2), soweit dies im Rahmen der Rücksetzausbreitung erforderlich ist, zu den entsprechenden *base/branch RPs* zurückgesetzt, die für den Fehlerfall zu den im ersten Schritt als fehlerhaft zurückgesetzten Recoveryblöcken (in Bild 6.1.2/1 wurde die Ausführung des Recoveryblockes $RB_{3,l+1}$ als fehlerhaft erkannt; zu $RB_{3,l+1}$ wurden in den Prozessen P_1 und P_2 die Wiederaufsetzpunkte $WP_{1,m}$ bzw. $WP_{2,n}$ als *branch RPs* in die Prozeßabläufe eingefügt) eingerichtet wurden.
Bild 6.1.2/2 zeigt den Systemzustand nach Rücksetzung von P_3 in der Situation von Bild 6.1.2/1 zum *base RP* des Recoveryblockes $RB_{3,l+1}$ und der anschließenden Rücksetzausbreitung.

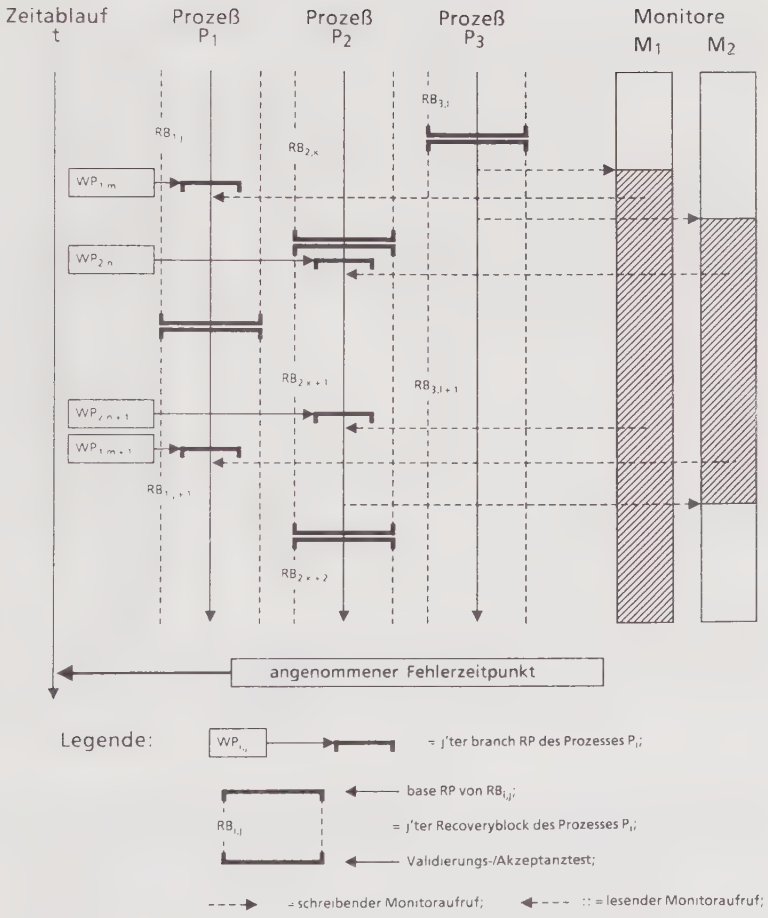
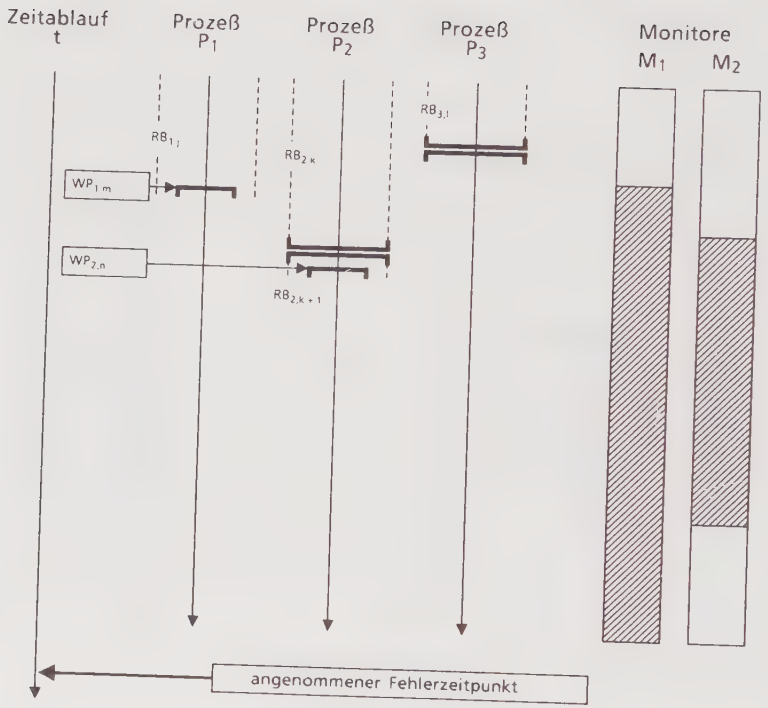


Bild 6.1.2/1: Systemzustand zum Fehlerzeitpunkt von P_3



Legende: siehe Bild 6.1.2/1.

Bild 6.1.2/2: Systemzustand nach Rücksetzung von P_3 zum base RP von $RB_{3,l+1}$

6.1.3. Wesentliche Ergebnisse

Ziel der Arbeiten [Kim78a], [Kim78b], [Kim78c] von K.H. Kim ist das Auffinden eines Schemas, welches die Implementierung einer benutzertransparenten Prozeßablauf-Sicherung durch Wiederaufsetzpunkte einhergehend mit deren effizienten Verwaltung ermöglicht, und das eine im Fehlerfall evtl. erforderliche Rücksetzausbreitung (mit der Gefahr des Dominoeffektes) begrenzt. Das wesentliche Ergebnis der Arbeiten von K.H. Kim bildet daher die Zusammenstellung von Regeln zum Anlegen und Freigeben von Wiederaufsetzpunkten (*branch RPs* und *base RPs*), die aufgrund einer intuitiven Vorstellung von Abhängigkeitsbeziehungen zwischen Prozessen entwickelt werden. Die Umsetzung der aufgestellten Regeln läuft dabei auf eine Monitor-Leseaufruf-Sicherung hinaus. Insofern bestehen vor allem Parallelen zu Abschnitt 5.1 der vorliegenden Arbeit, in dem die von K.H. Kim zur Vermeidung/Verhinderung des Dominoeffektes eingeschlagene Vorgehensweise als wirksam bestätigt wird.

Durch eine Beweisskizze wird versucht den Nachweis der folgenden Behauptung zu erbringen:

Satz 6.1.3/1. Bei Einhaltung der abgeleiteten Regeln müssen zu jedem Zeitpunkt genau diejenigen Wiederaufsetzpunkte zu den Prozessen eines Prozeßsystems verwaltet werden, deren Existenz unbedingt erforderlich ist, um im Fehlerfall in der Lage zu sein, das gegebene Prozeßsystem in den aktuellsten (noch) gültigen Systemzustand zu versetzen.

Bemerkung. Was in [Kim78a], [Kim78b], [Kim78c] nicht gesagt wird, aber beachtet werden muß, ist die Tatsache, daß der eben erwähnte Satz selbstverständlich nur unter Zugrundelegung von Kim's Fehlerbehandlungs-Modells (oder eines anderen dazu gleichwertigen Modells) seine Aussagekraft behält.
Der Begriff (*noch*) *gültiger Zustand* wird von Kim nicht explizit definiert.

Breiten Raum nimmt in [Kim78a], [Kim78b], [Kim78c] auch die Diskussion einer möglichen Implementierung der vorgeschlagenen Regeln zur benutzertransparenten Prozeßablauf-Sicherung und Wiederaufsetzpunkt-Verwaltung ein. Des weiteren werden auch Strategien erläutert, die eine zusätzliche Reduktion der Zahl aufzubewahrender Sicherungspunkte unter bewußter in Kaufnahme eines evtl. großen Rücksetzaufwandes ermöglichen. Grundlage ist hier die Überlegung, daß die Wahrscheinlichkeit der Rücksetzung eines Prozesses zu einem bestimmten Wiederaufsetzpunkt im Zeitverlauf abnimmt.

Zu erwähnen bleibt schließlich noch die Erörterung zweier dem Deadlock-Problem verwandten Erscheinungen, die bei der Fehlerbehandlung auftreten können und die sich am besten anhand der folgenden Bilder 6.1.3/1 und 2 veranschaulichen lassen.

Das erste Problem läßt sich anhand von Bild 6.1.3/1 wie folgt beschreiben: Angenommen Prozeß P_i gibt über einen Schreibaufruf im Recoveryblock $RB_{i,j}$ fehlerhafte Information an Prozeß P_k weiter, die dieser im Recoveryblock $RB_{k,l}$ entgegennimmt.
Besteht in dieser Situation zwischen den Prozessen P_i und P_k die folgende Abhängigkeit: Prozeß P_i wird in $RB_{i,j}$ durch ein *wait* blockiert und erst in $RB_{k,l+1}$ durch P_k mittels einer *signal*-Operation aktiviert, so muß, falls aufgrund der durch P_i übergebenen, fehlerhaften Information der zu $RB_{k,l}$ gehörende Akzeptanztest nicht positiv abgeschlossen werden kann, Prozeß P_k immer wieder zurückgesetzt werden, ohne daß eine Chance auf Korrektur des Fehlers besteht, denn P_i ist blockiert und würde erst im $RB_{k,l+1}$ aktiviert werden.
In dieser Situation sieht Kim zwei Möglichkeiten der Lösung: Die Zulassung der S-Propagation (d.h. der auf gegenseitigem Mißtrauen zwischen Prozessen beruhenden Rücksetzausbreitung) in bestimmten Situationen oder die Einführung der Regel, daß jeder Prozeß, der nicht innerhalb einer bestimmten Zeitspanne aktiv war, als fehlerhaft zurückzusetzen ist.

Das zweite Problem kann wie folgt skizziert werden: Angenommen in Bild 6.1.3/2 ist die von Prozeß P_i in $RB_{i,j}$ an $RB_{k,l}$ mittels des Monitors M übergebene Information fehlerhaft und Prozeß P_k wird deshalb durch den zu $RB_{k,l}$ gehörenden Akzeptanztest als fehlerhaft klassifiziert und zurückgesetzt, so bedeutet dies, daß aufgrund der R-Propagation auch Prozeß P_i zurückgesetzt werden muß.
Das heißt unter bestimmten Bedingungen und Abhängigkeitsverhältnissen zwischen Prozessen (in Bild 6.1.3/2 zwischen den Prozessen P_i und P_k) kann es zu einer Situation kommen, in der ein fehlerhafter Prozeß (in Bild 6.1.3/2 ist dies Prozeß P_i), obwohl er immer wieder aktiv wird, nie in die Lage versetzt wird, seinen Fehler zu korrigieren, da er immer wieder zurückgesetzt wird, bevor er einen entsprechenden Akzeptanztest (in Bild 6.1.3/2 den Akzeptanztest zu $RB_{i,j}$) durchlaufen kann, der den Fehler aufdeckt.
Eine Lösung dieses Problems erscheint Kim nur möglich, falls sichergestellt wird, daß ein Prozeß als Reaktion auf häufiges Zurücksetzen auch dann alternative Berechnungen durchführt, wenn er nicht als Fehlerverursacher identifiziert wurde.

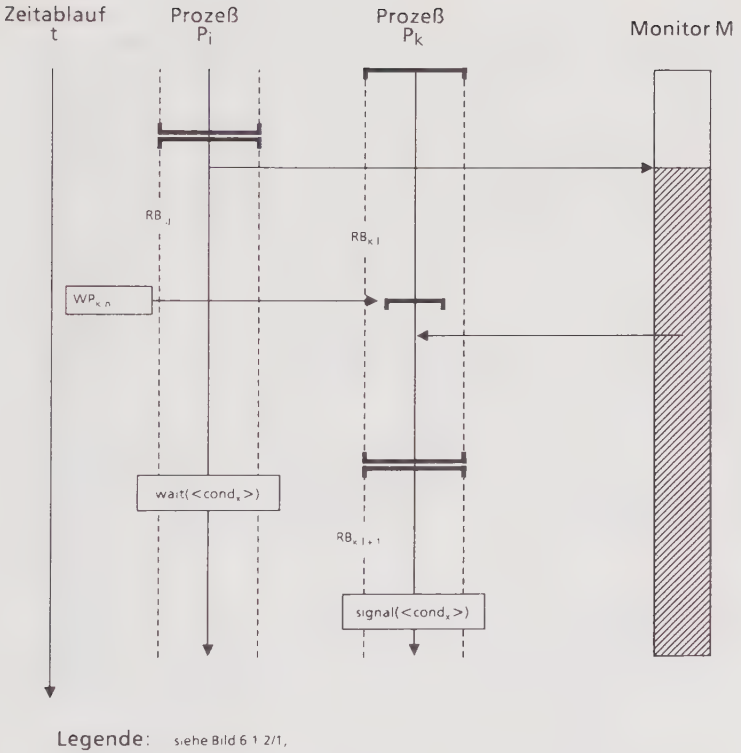
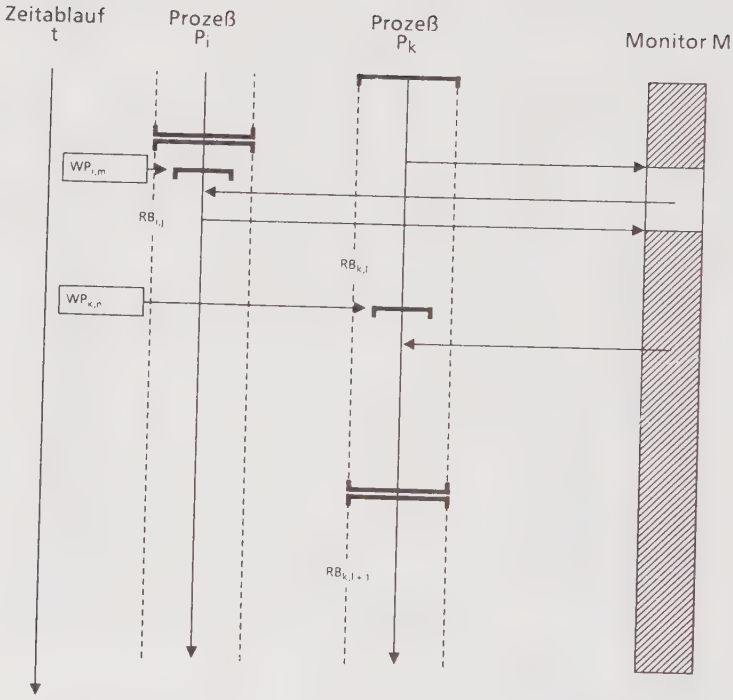


Bild 6.1.3/1: Veranschaulichung der Deadlock-Gefahr bei der Fehlerbehandlung (1)



Legende: siehe Bild 6.1.2/1,

Bild 6.1.3/2: Veranschaulichung der Deadlock-Gefahr bei der Fehlerbehandlung (2)

6.1.4. Ergänzende Anmerkungen

Kim's Beiträge stellen eine übersichtliche Einführung in den Problemkreis *dominoeffektfreie Fehlerbehandlung in Prozeßsystemen* dar. Wertvolle Einsichten und Anregungen geben sie insbesondere demjenigen, der sich mit der Realisierung von Systemen mit dieser Eigenschaft beschäftigt, wobei sich Kim in seinen Arbeiten allerdings auf die Beschreibung einer einzigen Verfahrensweise beschränkt und Implementierungsfragen ausschließlich vor dem Hintergrund des Recoveryblock- und Monitor-Konzeptes als dem zugrundeliegenden Software-Fehlertoleranz- bzw. Kommunikations-Mechanismus diskutiert werden.
In [Kim78c] findet man eine entsprechende *Pascal*-Spezifikation des zur Realisierung vorgeschlagenen Schemas.

Formalen Aspekten können die Arbeiten von K.H. Kim nur zum Teil genügen. Dies hat zwei Gründe: Zum einen fehlt u.a. eine Präzisierung der zentralen Begriffe gültiger (konsistenter) Prozeßsystem-Zustand und Dominoeffekt; zum anderen wird lediglich an zwei Stellen versucht, Behauptungen zu beweisen, und dies in einer nur sehr informellen und skizzenhaften Art, auf Zusammenhänge zwischen Prozeßsystem- und Fehlerbehandlungs-Modell wird dabei nicht eingegangen.
Trotz dieser Schwächen im formalen Bereich, erscheinen Kim's Beiträge aber insgesamt gesehen als sehr nützlich, und nicht zuletzt wegen der übersichtlichen Darstellung als gut in den aufgezeigten Problemkreis einführend und daher auch allgemein gesehen lesenswert.

Will man die von K.H. Kim in den zitierten Arbeiten diskutierte Dominoeffekt-Vermeidungs-/-Verhinderungs-Strategie, trotz des fehlenden Formalisierungsgrades, mit den im Rahmen der vorliegenden Arbeit abgeleiteten Aussagen in Beziehung setzen, so ist am ehesten eine Zuordnung zu dem Verfahren der Leseaufrufsicherung des Abschnittes 5.1 möglich.
Weitere Gemeinsamkeiten, wie auch zu der nachfolgend skizzierten Arbeit [Kan81] von K. Kant, bestehen in der Unterscheidung der Fehlerbehandlung mit und ohne Berücksichtigung potentieller Fehlerverursacher in Form der S-Propagation bzw. der R-Propagation.

6.2. Kant's Read-/Write-Modell

Bei K. Kant, [Kan81], stehen Überlegungen zur Dominoeffekt-Vermeidung/-Verhinderung unter Berücksichtigung potentieller Fehlerverursacher bei der Fehlerbehandlung im Mittelpunkt.
Seinen Untersuchungen legt er das folgende Prozeßsystem-Modell zugrunde.

6.2.1. Kant's Prozeßsystem-Modell

K. Kant geht bei seinem Prozeßsystem-Modell in [Kan81] davon aus, daß jeder Prozeß aus einer Anzahl von Recoveryblöcken aufgebaut ist. Die Kommunikation zwischen Prozessen und Synchronisation von Prozeßabläufen erfolgt über Monitore. Den Prozessen gemeinsame Daten werden innerhalb der Monitore verwaltet und auch ausschließlich über sie ausgetauscht. Dabei werden Monitoraufrufe in *read*, *write* und *update* (*read* + *write*) Aufrufe unterteilt. *Update*-Aufrufe sind jedoch in Kant's Modell nur auf lokal verifizierbaren Variablen (hierzu siehe den nächsten Abschnitt) erlaubt. Dadurch wird, eine entsprechende Anordnung der *read-/write*-Aufrufe innerhalb eines Recoveryblockes der Form $[(write,)^* (read,)^*]$ vorausgesetzt, erreicht, daß ein Prozeß lokal nicht verifizierbare Variablen einem Akzeptanztest unterziehen muß, bevor er sie an andere Prozesse weitergibt.

Den Untersuchungen von K. Kant ist folgende Struktur eines Monitorprozedur-Rumpfes zugrundegelegt:

begin
<nichtunterbrechbare Aktionenfolge 1>
if <Wartebedingung erfüllt> *then wait*(<Wartebedingung>);
<nichtunterbrechbare Aktionenfolge 2>
signal(<Wartebedingung>);
end;

Dabei ist angenommen, daß die nichtunterbrechbaren Aktionenfolgen 1 und 2 durch Recoveryblöcke repräsentiert werden. Mit nichtunterbrechbar ist hier gemeint, daß sich ein Prozeß innerhalb der Aktionenfolgen 1 und 2 durch ein *wait* nicht blockieren darf. Der Zugriff auf Daten unter gegenseitigem Ausschluß wird ansonsten durch das Monitorkonzept gewährleistet.
Zur Erläuterung der Begriffe Recoveryblock und Monitor sei auf die in 6.1.1 zitierte Literatur verwiesen.

6.2.2. Kant's Fehlerbehandlungs-Modell

Kant's Fehlerbehandlungs-Modell sieht wie folgt aus: Zu Beginn eines jeden Recoveryblockes wird automatisch ein Wiederaufsetzpunkt eingerichtet, der sich je nach dem, ob es sich um einen Recoveryblock innerhalb eines Prozesses oder Monitors handelt, auf die lokalen Daten eines Prozesses bzw. die durch einen Monitor verwalteten Daten bezieht. Ein derartiger Wiederaufsetzpunkt wird in [Kan81] als ein *User defined Checkpoint* (kurz *UDCP*) bezeichnet im Unterschied zu einem sogenannten *System defined Checkpoint* (kurz: *SDCP*), der zur Laufzeit immer dann in einen Prozeßablauf vor einen Monitoraufruf eines Prozesses P_i eingefügt wird, wenn kein *UDCP* vor diesem Monitoraufruf in P_i bereits eingerichtet wurde und wenn es sich um den zeitlich ersten Monitoraufruf des Prozesses P_i bzgl. des Monitors M nach einer Änderung des Monitorzustandes M durch einen anderen Prozeß P_j, $j \neq i$, handelt.
*UDCP*s und *SDCP*s beziehen sich immer nur auf die lokalen Daten eines Prozesses.

Fehlerbehandlung erfolgt bei Kant in drei Schritten: Im ersten Schritt werden alle fehlerhaften Prozesse zum nächstgelegenen, den Fehler korrigierenden *UDCP* zurückgesetzt. In Schritt zwei werden die lokalen Zustände rückgesetzter und nichtrückgesetzter Prozesse unter Einbeziehung vorhandener *SDCP*s aufeinander abgestimmt. Ergebnis ist ein bis auf die Monitorzustände konsistenter Prozeßsystem-Zustand. Die Monitorzustände müssen deshalb im dritten Schritt jeweils in den aktuellsten Zustand zurückgesetzt werden, der selbst und dessen Vorgänger durch keinen der im Zuge der Fehlerbehandlung zurückgesetzten Monitoraufrufe verändert wurden.
Im Rahmen der in den Schritten zwei und drei beschriebenen Fehlerbehandlung ist Rücksetzausbreitung vor allem durch Entzug von durch andere Prozesse bereits gelesener Information, aber zum Teil auch durch Entzug von durch nichtfehlerhafte Prozesse aktualisierter, evtl. gar nicht gelesener Information bedingt. Da Kant in seinem Modell die Fehlerbehandlung auch auf Erzeuger potentiell fehlerhafter Information ausdehnt, kann Rücksetzuasbreitung bereits im ersten Fehlerbehandlungsschritt erforderlich werden, wenn, ausgehend von einem als fehlerhaft erkannten Prozeß, die Menge der potentiellen Fehlerverursacher zu bestimmen ist.
Zur Ermittlung der potentiellen Verursacher eines Fehlers werden ***Akzeptanztests*** benutzt. Diese werden jeweils vor dem Verlassen eines Recovery-Blockes ausgeführt. Sie befinden über den Erfolg der Ausführung einer durchlaufenen Recovery-Block-Alternative.

In bezug auf die Überprüfbarkeit der Korrektheit von Daten durch Akzeptanztests unterscheidet Kant zwei Typen von Variablen. Über die Gültigkeit der Werte der *lokal verifizierbaren Variablen* kann die sie verwaltende Instanz (Prozeß oder Monitor) selbst entscheiden. Die Verifikation der Werte der *lokal nicht verifizierbaren Variablen*, diese werden ausschließlich durch Monitorprozeduren verwaltet und zwar nur durch

Operationen innerhalb der nichtunterbrechbaren Aktionenfolge 2 einer Monitorprozedur, ist übergeordneten Instanzen vorbehalten.
Diese Unterscheidung ermöglicht zwei Arten der Fehlerausbreitung zu betrachten: Fehlerausbreitung, die durch Akzeptanztests rechtzeitig unterbunden werden kann (*Fehlerausbreitung über lokal verifizierbare Variablen*), und Fehlerausbreitung, die erst verspätet eingedämmt werden kann (*Fehlerausbreitung über lokal nicht verifizierbare Variablen*). Die letztgenannte Form wird anhand von Bild 6.2.2/1 skizziert und erläutert (zu der im Bild verwendeten Notation vgl. die Legende zu Bild 6.1.2/1).

Angenommen der Akzeptanztest $AT_{3,k}$ deckt einen Fehler in Prozeß P_3 auf. Dieser Fehler kann durch P_3 selbst oder durch Übergabe fehlerhafter Information an P_3 (über Monitor M_1) durch die Prozesse P_1 oder/und P_2, trotz der Akzeptanztests $AT_{1,i}$ und $AT_{2,j}$, verursacht worden sein. In P_2 kann wiederum fehlerhafte Information von Prozeß P_4 aus, trotz $AT_{4,l}$, (über Monitor M_2) eingesickert sein.

Die skizzierte Fehlerausbreitung kann allerdings nur dann auftreten und muß deshalb nur dann berücksichtigt werden, wenn die in Bild 6.2.2/1 skizzierten Schreibaufrufe der Prozesse P_1, P_2 und P_4 lokal nicht verifizierbare Variablen der Monitore M_1 bzw. M_2 verändert haben. Unter der Annahme, daß Akzeptanztests innerhalb eines Prozeßsystems so aufeinander abgestimmt sind, daß sich ein Fehler nur über eine begrenzte Zahl von Kommunikationsschritten zwischen Prozessen fortpflanzen kann, bevor er entdeckt wird, läßt sich die Zahl der potentiellen Fehlerverursacher ermitteln und diese Art der Fehlerausbreitung eindämmen.
Können sich Fehler beispielsweise nur über einen Kommunikationsschritt ausbreiten, so bedeutet dies in der Situation des Bildes 6.2.2/1, dass fehlerhafte Information von Prozeß P_4 über P_2 nicht in P_3 einsickern kann. P_4 kommt damit als potentieller Verursacher des Fehlers von P_3 nicht in Frage.
Bei den Untersuchungen in [Kan81] ist ein *worst case error propagation limit* (kurz: *WCEPL*) von einem Kommunikationsschritt angenommen.

Fehlerausbreitung über die lokal nicht verifizierbaren Variablen ließe sich in Bild 6.2.2/1 auch unter der Annahme, Prozeß P_4 besäße die Fähigkeit lokal nicht verifizierbare Variablen des Monitors M_1 als übergeordnete Instanz zu validieren, begrenzen. In diesem Falle würden sowohl die Prozesse P_1 und P_2, als auch Prozeß P_4 als mögliche Fehlerverursacher aufgrund der erfolgreichen Validierung des Zustandes von Monitor M_1 durch $AT_{4,l}$ ausscheiden.

Über lokal verifizierbare Variablen kann Fehlerausbreitung unter den in [Kan81] gemachten Annahmen nicht erfolgen. Denn die Fehlerhaftigkeit der lokal verifizierbaren Variablen (zweiter Typ einer möglichen Fehlerausbreitung) wird nach

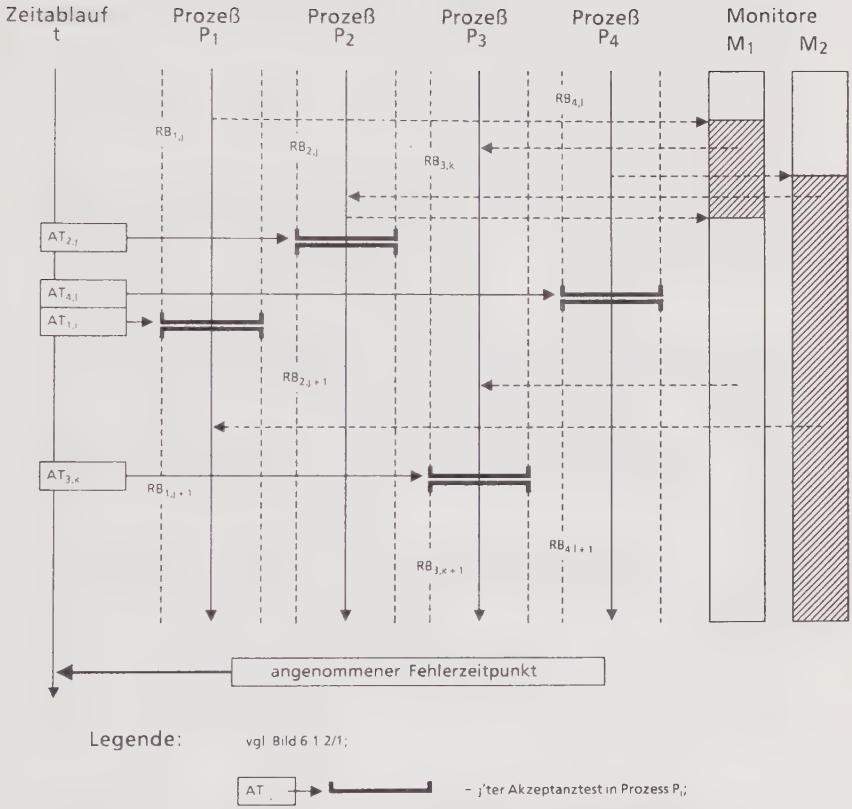


Bild 6.2.2/1: Veranschaulichung einer möglichen Fehlerausbreitung

Annahme rechtzeitig vor der Übergabe an andere Prozesse durch die innerhalb einer Monitorprozedur angesiedelten Akzeptanztests aufgedeckt und so eine Fehlerausbreitung a priori verhindert.

6.2.3. Wesentliche Ergebnisse

Kant versucht zu zeigen, daß für jedes Prozeßsystem P, das den folgenden sieben Bedingungen (B1) bis (B7) genügt, gilt (die zur Wiedergabe der Ergebnisse verwendete Notation entspricht nicht der in [Kan81] benutzten, sondern der in dieser Arbeit eingeführten;):

Satz 6.2.3/1: Sei P ein Prozeßsystem, das den Einschränkungen des Kant'schen Prozeßsystem-Modells genügt, und α sei eine aktive Berechnungsfolge von P. Ist der Zustand $val_P(\alpha)$ fehlerhaft, dann gilt: Es existiert ein Prozeß P_i von P, der von $val_P(\alpha)$ ausgehend in maximal zwei Rücksetzschritten in einen in bezug auf P_i konsistenten Zustand $val_P(\alpha_j)$, $j<|\alpha|$, zurückgesetzt werden kann, wobei kein Monitoraufruf eines anderen Prozesses P_k, der zeitlich in der Folge α vor $t(\alpha[j])$ lag, zurückgesetzt werden muß, um $val_P(\alpha)$ in einen in bezug auf das gesamte Prozeßsystem P konsistenten Zustand zu überführen.

Ergänzung: Als ersten Rücksetzschritt ("*Rollback*") bezeichnet K. Kant die Rücksetzung fehlerhafter Prozesse zu ihren aktuellsten, gesicherten und fehlerfreien Prozeßzuständen. Die weiteren Rücksetzschritte dienen der Ermittlung eines fehlerfreien Gesamtzustandes.

Die sieben Bedingungen, die erfüllt sein müssen, damit der genannte Satz Gültigkeit erlangt, lauten.

(B1) *Fehlerhafte Information* kann von einem Prozeß zu einem anderen übertragen werden, *wird* aber *spätestens nach maximal n Schritten als fehlerhaft erkannt*. Fehlerfortpflanzung endet also spätestens nach n Schritten. n wird als *Worst Case Error Propagation Limit* (*WCEPL*) bezeichnet. Bei Kant's Beweis des wiedergegebenen Satzes wird $WCEPL ::= 1$ vorausgesetzt.

(B2) *Update-Aufrufe* auf lokal nicht verifizierbaren Variablen eines Monitors *sind unzulässig*.

(B3) *Innerhalb eines Recoveryblockes müssen* alle *write-Aufrufe*, die lokal nicht verifizierbare Variablen eines Monitors ändern, *read-Aufrufen* innerhalb dieses Recoveryblockes *zeitlich vorangehen*.
Durch die Bedingungen (B2) und (B3) wird erreicht, daß Prozesse gelesene,

innerhalb von Monitoren lokal nicht verifizierbare Information in Akzeptanztests überprüfen müssen, bevor sie diese weitergeben.

(B4) *Recoveryblöcke dürfen nur aus einer beschränkten Anzahl von Monitoraufrufen aufgebaut sein* oder weniger restriktiv. *Ein aktivierter Recoveryblock darf während seiner Ausführung nur eine vorab bekannte maximale Anzahl von Monitoraufrufen absetzen.*

(B5) *Ein Prozeß darf nicht eine beliebige Zeitdauer lang blockiert sein.* Jeder Recoveryblock eines Prozesses muß nach einer a priori abschätzbaren Zeitspanne abgeschlossen und der zugehörige Akzeptanztest durchlaufen sein.

(B6) Die *globale Gültigkeit von Variablenwerten muß* durch Vergleich mit einer festen Anzahl von Werten *nachweisbar sein.*

(B7) *Ein Wiederaufsetzpunkt (SDCP) muß* vor einem Aufruf eines Prozesses P_i bzgl. eines Monitors M immer *dann eingerichtet werden, wenn* unmittelbar vor diesem Aufruf *ein UDCP zu* P_i *nicht existiert und der Zustand von* M seit dem letzten Aufruf von P_i bzgl. M durch einen anderen Prozeß P_j, $j \neq i$, *verändert wurde.*

Der Beweis des zitierten Satzes gelingt allerdings in [Kan81] nicht überzeugend. Zum einen werden die Hilfssätze, unter deren Benutzung der Nachweis geführt wird, nicht nur unzureichend bewiesen, sondern auch bereits mißverständlich und ungenau formuliert, was unter anderem daran liegt, daß einige wichtige Begriffe wie z.B. Dominoeffekt, konsistenter Prozeßsystem-Zustand usw. nicht explizit definiert werden. Zum anderen wird beim Nachweis der Gültigkeit aufgestellter Behauptungen auf die von einem Prozeßsystem zu fordernden Eigenschaften (B1) bis (B7) nicht zurückgegriffen. Letzteres erschwert unter anderem die Begründung der Bedingungen (B1) bis (B7) als *notwendig und hinreichend*, die in [Kan81] lediglich auf eine sehr informelle, eher motivierende als die Erforderlichkeit aufzeigende Art und Weise geführt wird. Detaillierte Angaben zu den Schwächen der Dissertation von K. Kant und Vorschläge zu ihrer Präzisierung wurden in [Bat83] zusammengestellt bzw. unterbreitet.

Neben dem skizzierten Hauptergebnis, werden in [Kan81] die folgenden Themen andiskutiert:

(1) Das *Problem der Verwaltung von Recovery-Information.* Bei diesem Punkt geht es um die Frage wie häufig Wiederaufsetzpunkte gesetzt und vor allem wie lange sie aufbewahrt werden müssen, um der ihnen zugedachten Aufgabe gerecht zu werden, wobei die in [Kan81] zu diesem Problemkreis gemachten Bemerkungen und aufgezeigten Lösungsmöglichkeiten im Vergleich zu anderen Arbeiten, wie z.B. [Kim78a], [Kim78b], [Kim78c] und [Rus76] einen geringeren Umfang einnehmen.

(2) *Möglichkeiten der Erweiterung des den Überlegungen von K. Kant zugrundeliegenden Monitor-Modells* z.B. um die Zulässigkeit geschachtelter

Monitoraufrufe; die Erläuterungen zu diesem Punkt sind informell und skizzenhaft gehalten

(3) *Auswirkungen der Annahme* $1<WCEPL\leq n$, $n\in\mathbb{N}$, *auf die Dominoeffektfreiheit* von Prozeßsystemen, welche die vorab aufgelisteten Bedingungen (B1) bis (B7) erfüllen; es zeigt sich, daß die Verallgemeinerung von *WCEPL* auf Werte größer 1 zu Problemen führt, die in [Kan81] aufgezeigt, jedoch nicht abschließend gelöst werden.

(4) Versuch der *Verallgemeinerung der Ergebnisse* durch Ersetzung des Kommunikations-/Synchronisations-Mechanismus Monitor durch andere Sprachkonstrukte, Kant's Überlegungen laufen in diesem Bereich auf die Simulation des Monitorkonzeptes hinaus Er legt seinen Untersuchungen jedoch keinen Simulationsbegriff (vgl. [Bat82], [Hof84]) zugrunde. Nicht zuletzt aus diesem Grunde werden die gewonnen Erkenntnisse nur grob beschrieben; sie werden in Form zu treffender Einschränkungen, die sicherstellen sollen, daß ein gegebenes, simulierendes System nur das Kant'sche Monitor-Modell und nichts darüber hinaus nachbildet, dargelegt.

6.2.4. Ergänzende Anmerkungen

Obwohl Kant's Arbeit in vielen Punkten ungenau ist und es letztlich dem Leser überlassen bleibt, sie entsprechend zu präzisieren, will er sich von der Korrektheit der gemachten Aussagen überzeugen, so enthält Kant's Dissertation doch einige Ansätze und Anregungen, die sie von anderen Veröffentlichungen in wesentlichen Punkten unterscheiden und deshalb erwähnenswert machen. In diesem Zusammenhang sind vor allem die getrennte Betrachtungsweise von *lokal verifizierbaren* und *lokal nicht verifizierbaren Variablen* (Daten), die *Berücksichtigung potentieller Fehlerverursacher* bei Zugrundelegung eines *Worst Case Error Propagation Limits von eins* (*WCEPL=1*) und *der Versuch die abgeleiteten Ergebnisse zu verallgemeinern*, d.h. über Monitore hinaus auch auf die Verwendung anderer Kommunikationsmechanismen zu übertragen, zu nennen Nicht zuletzt diese Ideen und die Ungenauigkeit ihrer Ausführung gaben den Anstoß zum Anfertigen der vorliegenden Dissertation.

Bezüge zu dem in [Kan81] abgeleiteten Hauptergebnis, wie es im vorangehenden Abschnitt skizziert wurde, enthalten die Abschnitte 4.2, 4.3, 4.4 und 5.1. In diesen wurde die Wichtigkeit von Kant's Bedingungen (B1) bis (B7) für die Gültigkeit des Satzes 6.2.3/1 vom Prinzip her formal nachgewiesen. Kant's Hauptergebnis konnte aber nicht nur bestätigt, sondern auch hinsichtlich des zugrundegelegten Prozeßsystem- und Fehlerbehandlungs-Modells in wesentlichen Punkten verallgemeinert werden.

Der interessierte Leser sei darauf hingewiesen, daß die Darstellung der wesentlichen Ergebnisse der Arbeit von K. Kant in einer kürzlich erschienenen

Veroffentlichung, [KS85], einen solideren Eindruck hinterläßt, als die Ausführungen in der zugehörigen und hier beschriebenen Dissertation [Kan81].

6.3. Russell's Erzeuger-/Verbraucher-Modell

D.L. Russell befaßt sich in [Rus76], [Rus77], [Rus80] mit dem Problem der dominoeffektfreien Fehlerbehandlung in Nachrichtensystemen. Diese Systeme sind durch das folgende Prozeßsystem-Modell charakterisiert.

6.3.1. Russell's Prozeßsystem-Modell

Zur Kommunikation und Synchronisation stehen Prozessen in dem von D.L. Russell in [Rus76], [Rus77] und [Rus80] seinen Untersuchungen zugrundegelegten Prozeßsystem-Modell atomare Operationen zum einfügen, *send*, und entfernen, *receive*, von Nachrichten in bzw. aus einem Puffer zur Verfügung. Diese *send*- und *receive*-Befehle ermöglichen es Prozessen Informationen über potentiell unendliche *FIFO-Puffer* auszutauschen.

6.3.2. Russell's Fehlerbehandlungs-Modell

Das wesentliche Merkmal des Russellschen Fehlerbehandlungs-Modells stellen die drei Operationen *mark*, *restore* und *purge* dar. Über *mark* kann der aktuelle Zustand eines Prozesses gesichert und ein Wiederaufsetzpunkt für diesen Prozeß angelegt werden, zu dem der Prozeß bei Bedarf durch *restore* zurückgesetzt werden kann. Mit Hilfe von *purge* können nicht mehr benötigte Wiederaufsetzpunkte gelöscht und der von ihnen belegte Speicherplatz freigegeben werden.

Da Nachrichten die einzigen Prozessen gemeinsamen Daten in einem Prozeßsystem sind, genügt es im Russellschen Fehlerbehandlungs-Modell zum Zwecke der Fehlerbehandlung neben den zu bestimmten Zeitpunkten jeweils zu sichernden Zuständen einzelner Prozesse nur noch den Nachrichtenfluß mitaufzuzeichnen. Dies geschieht in Form der Aufzeichnung einer Nachrichtenhistorie, die zu jedem Puffer geführt wird. In ihr wird vermerkt, wer welche Information (in welchem Prozeßabschnitt) im Nachrichtenpuffer hinterlegt oder aus ihm entnommen hat.
Im Fehlerfall wird der den Fehler verursachende Prozeß zum aktuellsten vor dem Fehlerzeitpunkt angelegten Wiederaufsetzpunkt zurückgesetzt. Alle seit dieser Zeit von dem Prozeß empfangenen Nachrichten werden in die betreffenden Puffer zurückgelegt, gesendete Informationen müssen aus den entsprechenden Puffern entnommen werden. Dies kann zur Rücksetzausbreitung auf andere Prozesse eines Prozeßsystems führen

Von D.L. Russell werden zwei Arten von Rücksetzausbreitung unterschieden. Solche, die auf den Entzug bereits empfangener Information zurückzuführen ist, in [Rus80] wird sie als *M-Propagation* bezeichnet, und solche, die durch Rückgabe empfangener Nachrichten verursacht wird, in [Rus80] *R-Propagation* genannt. Der zweite Typ ist nur dann zu beachten, wenn die Reihenfolge, in der einzelne Nachrichten aus einem Puffer entnommen und ausgewertet werden, von Bedeutung ist. Dies ist z.B. dann der Fall, wenn ein Prozeßsystem eine FIFO-Verarbeitung der in einen Puffer eingefügten Nachrichten gewährleisten soll.

In der Situation von Bild 6.3.2/1 führt beispielsweise die Rücksetzung von Prozeß P_3 zum Wiederaufsetzpunkt $WP_{3,i+1}$ zum Entzug der Nachrichten N_{i+2}, N_{i+3} und damit zu einer M-Propagation, welche die Rücksetzung von P_1 und P_2 zu den Wiederaufsetzpunkten $WP_{1,j+1}$ bzw. $WP_{2,k+1}$ erforderlich macht.

Die Rückgabe einer Nachricht, wie z.B. N_{i+2}, kann, wie Bild 6.3.2/2 zeigt, ebenfalls zur Rücksetzausbreitung (R-Propagation) führen, wenn z.B. gefordert wird, daß Nachrichten in einer *FIFO-Reihenfolge* aus dem Puffer *PU* entnommen und weiterverarbeitet werden.

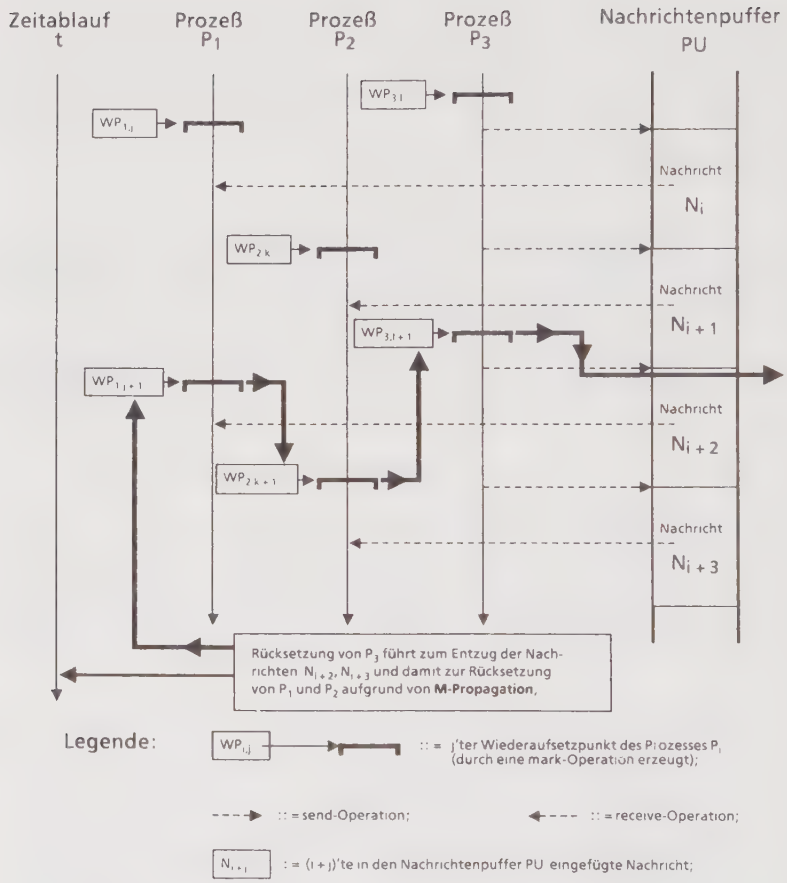


Bild 6.3.2/1: Beispiel einer M-Propagation bei Entzug von N_{i+2} und N_{i+3}

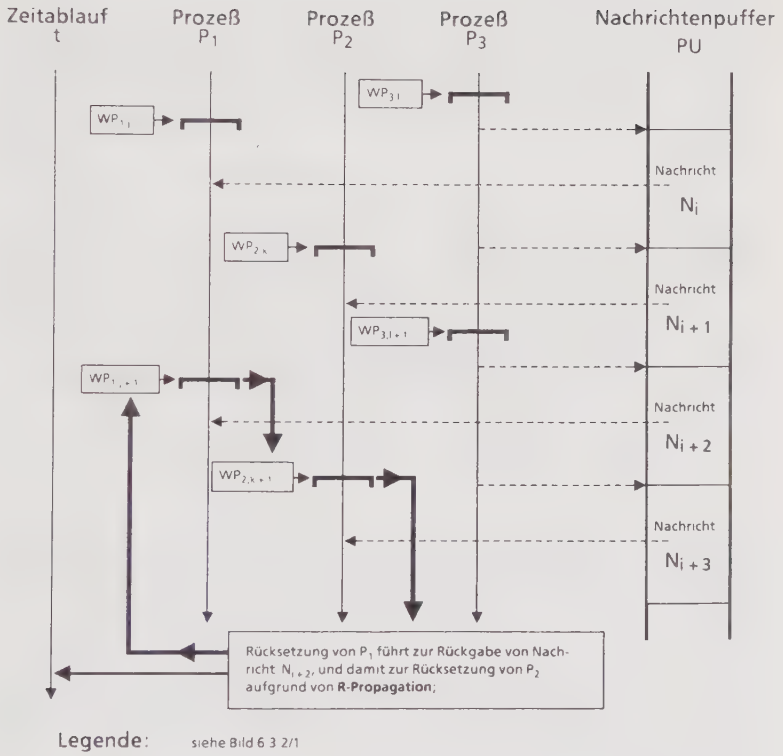


Bild 6.3.2/2: Beispiel einer R-Propagation bei Rückgabe von N_{i+2}

6.3.3. Wesentliche Ergebnisse

Zu den wichtigsten Resultaten die im Rahmen der Arbeiten [Rus76], [Rus77] und [Rus80] von D L Russell abgeleitet und bewiesen werden, zahlen die folgenden Satze, die samtlich in [Rus80] veröffentlicht wurden (die im folgenden zur Darstellung gewählte Notation ist entsprechend an die in diesem Beitrag eingeführte Schreibweise angepaßt):

Satz 6.3.3/1: Genügt ein Prozeßsystem P, in dem die Kommunikation zwischen Prozessen ausschließlich auf der Basis von (auf Nachrichtenpuffern ausführbaren) *send-/receive*-Operationen geführt wird einer der beiden folgenden Bedingungen (1) oder (2):

(1) die Entnahmereihenfolge von Nachrichten aus den Nachrichtenpuffern von P ist für die Korrektheit der Berechnungsabläufe von P bedeutungslos;

(2) für die Nachrichten eines jeden Nachrichtenpuffers von P gibt es nur einen Empfänger;

so kann bei einer evtl. während einer aktiven Berechnung in P erforderlich werdenden Fehlerbehandlung R-Propagation nicht auftreten.

Satz 6.3.3/2 Zu jedem Prozeßsystem P, in dem R-Propagation während einer Fehlerbehandlung auftreten kann, läßt sich ein Prozeßsystem konstruieren, das sich äquivalent zu P verhält (eine genaue Definition von äquivalent fehlt in [Rus80], aus dem Zusammenhang wird aber deutlich, daß es sich hierbei um eine Simulation, die vollständig und sicher im Sinne von [Lip74], [Bat82], [Hof84] ist, handeln muß

Satz 6.3.3/3. Ist P ein Prozeßsystem, in dem R-Propagation nicht auftritt und in dessen aktiven Berechnungsabläufen zyklische Kommunikationsbeziehungen zwischen Prozessen nicht vorkommen, so kann Fehlerbehandlung in P nicht zum Dominoeffekt (im Sinne der Definition von D L Russell einer sich zyklisch von Prozeß zu Prozeß fortpflanzenden Rücksetzausbreitung) führen.

Satz 6.3.3/4 Sei P ein aus m Prozessen bestehendes System, in welchem R-Propagation nicht auftreten kann. Jeder Prozeß sei aus Recoveryblöcken aufgebaut, die bzgl. der Ausführungsreihenfolge von *send-/receive*-Operationen die Struktur (*mark; (receive;)* (send;)**) besitzen. Dann ist P dominoeffektfrei.
Kann des weiteren in der vorgegebenen Situation für die Anzahl der zur Ausführungszeit innerhalb eines Recoveryblockes abgesetzten *receive*-Aufrufe eine obere Schranke n angegeben werden, so läßt sich zusätzlich zeigen, daß Fehlerkorrektur im Fehlerfall die Zurücksetzung von maximal $(m-1)*(n-1)$ *receive*-Operationen erfordert, deren Rücksetzung selbst durch Einführung zusätzlicher Wiederaufsetzpunkte nicht vermieden werden könnte.

Die bisher skizzierten Sätze stellen nur einen Teil der von D.L. Russell abgeleiteten Ergebnisse dar. Weitere Aussagen der skizzierten Art inklusive der zugehörigen Beweise findet man in [Rus76], wo neben einer *Pascal*-Spezifikation des Fehlerbehandlungs-Modells für R-propagationsfreie Systeme auch das Problem der Freigabe von Wiederaufsetzpunkten im Rahmen einer formalen Untersuchung behandelt wird.

6.3.4. Ergänzende Anmerkungen

Mit den beschriebenen Basiselementen des Russellschen Modells lassen sich Probleme der Fehlerbehandlung in nachrichtenaustauschenden Prozeßsystemen anschaulich aufzeigen und Lösungen für die Begrenzung der Rücksetzausbreitung in derartigen Systemen sowie für die Konstruktion dominoeffektfreier Systeme finden.
Da die von D.L. Russell in [Rus76], [Rus77] und [Rus80] abgeleiteten Ergebnisse im Vergleich zu ähnlichen Aussagen in den vorab vorgestellten Arbeiten von K.H. Kim und K. Kant ausreichend präzise bewiesen werden, kommen die Beiträge von D.L. Russell, trotz der Einschränkung auf nachrichtenaustauschende Prozeßsysteme und der Nichtberücksichtigung potentieller Fehlerverursacher in seinem Modell, den Zielen und Vorstellungen dieser Dissertation von allen erwähnten Arbeiten am nächsten.

Hinsichtlich der Vergleichbarkeit der in den zitierten Arbeiten von D.L. Russell zum Thema Dominoeffekt abgeleiteten Ergebnisse mit denjenigen der vorliegenden Ausarbeitung läßt sich feststellen: Analog zu [Rus80], erwies es sich auch im Rahmen der vorliegenden Dissertation als sinnvoll, vermeidbare und unvermeidbare Zurücksetzung von Aktionen zu unterscheiden. Des weiteren wurde die Aussage von Satz 6.3.3/1 in Abschnitt 3.4.4 in einem globaleren Zusammenhang indirekt bestätigt, und die Aussagen der Sätze 6.3.3/3 und 4 konnten in den Abschnitten 5.5 bzw. 5.1 auf den Fall allgemeiner Prozeßsysteme übertragen und vertieft werden.

7. Zusammenfassung mit Schlußbemerkungen

Wie einleitend erwähnt, lag der Schwerpunkt dieser Arbeit auf der formalen Untersuchung des bei der Fehlerbehandlung in Prozeßsystemen auftretenden Phänomens Dominoeffekt.
Eine Gegenüberstellung der im Rahmen dieser Abhandlung abgeleiteten Aussagen zu vorangehend beschriebenen, wesentlichen Ergebnissen veröffentlichter Beiträge von D.L. Russell, K.H. Kim und K. Kant, diese Autoren haben sich nach dem gewonnen Eindruck und Kenntnisstand am weitaus intensivsten mit dem Phänomen Dominoeffekt beschäftigt, ohne mit dieser Behauptung Abhandlungen von T. Anderson, R. Kerr, J.C. Knight, P.A. Lee, E. Best, B. Randell, J.J. Horning, S.K. Shrivastava, J. Banatre und K. Weber (vgl. [AK76], [AK81], [AL81], [BR81], [HR73], [Hor74], [SB78] bzw. [Web83]) schmälern zu wollen, zeigt, daß in der vorliegenden Ausarbeitung bestehende Erkenntnisse auf der Basis eines in Verbindung mit der Fehlerbehandlung eingeführten formalen Prozeßsystem-Modells bestätigt, in wichtigen Punkten verallgemeinert und zusammen mit neu hinzukommenden Resultaten präzise bewiesen wurden.
Die Ableitung der in den Kapiteln drei, vier und fünf enthaltenen Kernaussagen dieser Arbeit erfolgte in mehreren Schritten und in einer Weise, die einen engen Bezug vom zugrundegelegten formalen Modell zur Realität unterstützt.

Nach einer ausführlichen Motivation und einer genauen Abgrenzung des Themas der vorliegenden Arbeit, wurden ein Prozeßsystem- und ein Fehlerbehandlungs-Modell beschrieben, die beide durch Abstraktion von realen Rechensystemen auf das für die im Rahmen dieser Dissertation verfolgten Ziele Wesentliche entstanden sind.
Dabei gingen die Überlegungen von einem logisch zentralisierten Rechensystem mit einem fehlerfrei funktionierenden Betriebssystemkern aus, das sowohl in Form eines Monoprozessorsystems, Mehrprozessorsystems, d.h. eines physisch zentralisierten Systems, als auch eines Mehrrechnersystems, d.h. eines physisch verteilten Systems, realisiert vorliegen kann. Verschiedene Annahmen zur Funktionsweise des Rechensystems, die sich im wesentlichen an in [LS81] und [Mos81] abgeleiteten Ergebnissen orientierten, ermöglichten eine genaue Modellierung des Verhaltens im Fehlerfall, die Eingrenzung der Betrachtungen auf zwei Fehlertypen, nämlich die Prozeßsystem- und die Rechensystemfehler, sowie die Gewinnung eines intuitiven Verständnisses für Fehlerbehandlungsmaßnahmen und die in diesem Zusammenhang einzuleitende Rücksetzung von Aktionen.
Letzteres war erforderlich, um eine an der Realität orientierte präzise Grundlage zur Gewinnung von Aussagen über Fehlerbehandlung in Prozeßsystemen und des dabei auftretenden Phänomens Dominoeffekt zu gewinnen.

Die formalen Untersuchungen hatten ihren Ausgangspunkt bei der Beschreibung der Menge der im Fehlerfall zurückzusetzenden Aktionen und der Charakterisierung der Menge der potentiellen Fehlerverursacher. Es wurde gezeigt, daß die Größe der

Menge der im Fehlerfall zurückzusetzenden Aktionen von der zugrundegelegten Recoveryfunktion, welche Eigenschaften des verwendeten Fehlerbehandlungsverfahrens formal beschreibt, abhängt. Welches Fehlerbehandlungsverfahren gewählt werden kann, hängt im Einzelfall vom Prozeßsystem ab.
Es wurden vier Prozeßsystemtypen, Systeme unabhängiger, koordinierender, kommunizierender und kooperierender Prozesse, sowie, in Anlehnung an [ALß1], zwei Klassen von Fehlerbehandlungsverfahren, Rückwärts- und Vorwärtsrecoveryverfahren, unterschieden.
Die zum Thema Dominoeffekt nachgewiesenen wesentlichen Ergebnisse wurden jedoch ohne Rückgriff auf einen bestimmten Prozeßsystemtyp abgeleitet, und auch bei der Fehlerbehandlung wurden keine besonderen, zur Entwicklung fehlertoleranter Software geeigneten Sprachkonstrukte, wie sie beispielsweise in [JKR76] diskutiert werden, vorausgesetzt.
Durch diese Vorgehensweise wurde die Übertragbarkeit der bewiesenen Aussagen gesichert.

Eine Präzisierung des Begriffes Dominoeffekt und die Unterscheidung vermeidbarer und unvermeidbarer Zurücksetzung ermöglichten, unter impliziter Zugrundelegung einer Rückwärtsrecovery repräsentierenden allgemeinen Fehlerbehandlungsfunktion τ_1, die stufenweise Ableitung hinreichender und notwendiger Bedingungen für das Nichtauftreten des Dominoeffekts sowohl bei Nichtberücksichtigung als auch bei Berücksichtigung potentieller Fehlerverursacher im Fehlerfall.
Auf der Basis der so ermittelten Ursachen des Dominoeffekts konnten schließendlich, eingeschränkt auf den Fall der Nichtberücksichtigung potentieller Fehlerverursacher, Beispiele verschiedener dominoeffektfreier Prozeßsystemklassen vorgestellt und zusammen mit verschiedenen Berechnungssicherungs-Strategien, deren Wirksamkeit im Hinblick auf Dominoeffektfreiheit nachgewiesen wurde, näher charakterisiert werden.

Gerade die zuletzt skizzierten Betrachtungen motivieren die folgende Überlegung zur Weiterarbeit:

Aufbauend auf den Erkenntnissen der vorliegenden Abhandlung, die sich nicht zuletzt aufgrund der Aufgabenstellung auf rein theoretische Betrachtungen beschränken mußte, erscheint es besonders wünschenswert und reizvoll, auch praktische Erfahrungen mit den vorgestellten Modellen, Strategien und Konzepten zu sammeln, wobei insbesondere die Überprüfung von Implementierungsaspekten und Effizienzuntersuchungen einen breiten Raum einnehmen sollten.
Als ideale Entwicklungs- und Experimentierumgebung erscheint in diesem Zusammenhang das von Bell Laboratories entwickelte, an Hochschulen und mittlerweile auch in der Industrie weit verbreitete Betriebssystem UNIX, mit all seinen Hilfsmitteln, wie etwa LEX und YACC, mit welchen, unter Rückgriff auf die mit UNIX untrennbar verwobene Programmiersprache C, auf relativ einfache Weise ein Interpreter für eine

Prozeßsystembeschreibungssprache entwickelt werden könnte, welche beispielsweise explizit zwischen lokalen und globalen Variablen unterscheidet sowie benutzertransparent eine automatische Zustandssicherung für Werte globaler Variablen realisiert und Operationen zum Einrichten von lokalen Wiederaufsetzpunkten bereitstellt.
Mit einem solchen Interpreter eng verbunden sollte ein Laufzeitsystem sein, das es ermöglicht, innerhalb eines Prozeßsystems aktive Berechnungen zu verfolgen, sie genau zu analysieren, das benutzerinitiiert oder von sich aus, als Analogon zu betriebssysteminitiiert, Wiederaufsetzpunkte setzen kann, und auch zur Simulation relevanter Fehlerfälle geeignet ist. Vergleichbar zum Deadlockproblem, könnten in solch einem Umfeld drei unterschiedliche Strategien zur Beseitigung des Dominoeffektes untersucht werden, die sich hinsichtlich ihrer wesentlichen Eigenschaften wie folgt umreißen lassen:

1. Bei der *Dominoeffektvermeidungs-Strategie* wird ein Prozeßsystem so strukturiert, daß im Fehlerfall kein Dominoeffekt auftreten kann (*"statische Dominoeffektverhinderung"*).

2. Bei der *Dominoeffektverhinderungs-Strategie* werden, abhängig vom Berechnungsverlauf, vom System her Wiederaufsetzpunkte (automatisch) eingefügt, die im Fehlerfall einen Dominoeffekt verhindern (*"dynamische Dominoeffektverhinderung"*)

3. Die *Dominoeffekterkennungs- und -beseitigungs-Strategie*, als dritte und letzte unterschiedene Alternative, basiert darauf, daß sie bei Erkennung der Dominoeffektgefahr zur Fehlerkorrektur Vorwärtsrecoveryverfahren einsetzt, um einen entdeckten Fehler zu beseitigen. Dies insbesondere dann, wenn die Dominoeffektgefahr unter Zugrundelegung einer Rückwartsrecovery-Strategie erkannt wurde.

Neben den bereits angesprochenen Effizienzbetrachtungen könnten des weiteren am Beispiel konkreter Anwendungsfälle Erfahrungen gewonnen werden, wie lange unter realen Bedingungen Sicherungsdaten aufbewahrt werden müssen, oder auf welche Weise ihr Wachstum, im Falle eines bewußten und explizit zu machenden Verzichts auf die Abdeckung aller theoretisch möglichen Widrigkeiten, auf ein praktisch vertretbares Maß reduziert werden kann.
Von ähnlichem Interesse wären schließlich praktische Erfahrungen zur Beantwortung der Frage, welches Fehlerbehandlungsverfahren, bezogen auf real gegebene Prozeßsysteme, am effizientesten einsetzbar ist.

Sowohl die Beantwortung der zuletzt aufgeworfenen Frage, als auch nähere Untersuchungen zu den zuvor für das weitere Vorgehen vorgeschlagenen Themenkomplexen, bleiben zukünftigen Aktivitäten auf dem Gebiet der Fehlerbehandlung in Prozeßsystemen vorbehalten.

Literaturverzeichnis

Die zur Erstellung der vorliegenden Abhandlung benutzte Literatur kann der folgenden, alphabetisch geordneten Referenzliste entnommen werden:

[AK76] T. Anderson, R. Kerr, "Recovery Blocks in Action: A System Supporting High Reliability", *Proc. of 2'nd Inter. Conf. on Software Engineering*, San Francisco, CA, Oktober 1976, S. 447-457

[AK81] T. Anderson, J.C. Knight, "Practical Software Fault Tolerance for Real Time Systems", University of Newcastle upon Tyne, Computing Laboratory, *Technical Report Series*, No. 169, 1981

[AL81] T. Anderson, P.A. Lee, "Fault Tolerance Principles and Practice", *Prentice-Hall*, 1981

[And79] D.M. Andrews, "Using Executable Assertions for Testing and Fault Tolerance", *Digest of Papers FTCS-9: Ninth Annual International Symposium on Fault-Tolerant Computing*, Madison, WI, Juni 1979, S. 102-105

[Avi85] A. Avizienis, "The N-Version Approach to Fault-Tolerant Software", *IEEE Transactions on Software Engineering*, vol. SE-11, No. 12, Dezember 1985, S. 1491-1501

[Bat82] P. Bathelt, "Vergleich von Synchronisationsmechanismen", *Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung*, Universität Erlangen-Nürnberg, Band 15, Nr. 3, 1982

[Bat83] P. Bathelt, "Ergebnisse einer Untersuchung der Arbeit 'Error Recovery in Concurrent Processes' von K. Kant", *Manuskript*, 1983

[BK81] M. Berg, I. Koren, "A Module Replacement Policy for Dynamic Redundancy Fault-tolerant Computing systems", *11'th IEEE Symposium on Fault-Tolerant Computing*, Portland, Maine, 1981, S.90-95

[BR81] E. Best, B. Randell, "A Formal Model of Atomicity in Asynchronous Systems", *Acta Informatica*, vol. 16, 1981, S. 93-124

[BS81] C. Bellon, G. Saucier, "Protection Against External errors in a Dedicated System", *11'th IEEE Symposium on Fault Tolerant Computing*, Portland, Maine, 1981, S. 122-128

[Cha75a] K.M. Chandy, "A Survey of Analytic Models of Roll-Back and Recovery Strategies", *Computer*, vol. 8, No. 5, Mai 1975, S. 40-47

[Cha75b] K.M. Chandy et al., "Analytic Models for Roll-Back and Recovery Strategies in Database Systems", *IEEE Transactions on Software Engineering*, vol. SE-1, No. 1, März 1975, S. 100-110

[CR72] K.M. Chandy, C.V. Ramamoorthy, "Rollback and Recovery Strategies for Computer Programs", *IEEE Transactions on Computers*, vol. C-21, No. 6, Juni 1972, S. 546-556

[Cri82] F Cristian, "Exception Handling and Software Fault Tolerance", *IEEE Transactions on Computers*, vol. C-31, No. 6, Juni 1982, S. 531-540

[Dao79] N.F. Daouk, "Optimal Checkpointing in Real Time Multiprogramming Systems", *Dissertation*, Juni 1979

[Den76] P.J. Denning, "Fault Tolerant Operating Systems", *Computing Surveys*, vol 8, No. 4, Dezember 1976, S.359-389

[Dhi82] B.S. Dhillon, "Software Reliability - Bibliography", *Microelectronics and Reliability*, vol. 22, No. 3, 1982, S. 625-640

[Dij68] E.W. Dijkstra, "Cooperating Sequential Processes", in *Programming Languages*, ed. F. Genuys, Academic Press London - New York, 1968, S. 43-112

[Fab73] R.S. Fabry, "Dynamic Verification of Operating System Decisions", *CACM*, vol. 16, No.11, November 1973, S. 659-668

[FGL82] M.J. Fischer, N.D. Griffeth, N.A. Lynch, "Global States of a Distributed System", *IEEE Transactions on Software Engineering*, vol. SE-8, No. 3, Mai 1982, S. 198-202

[FLP82] M.J. Fischer, N.A. Lynch, M.S. Paterson, "Impossibility of Distributed Consensus with one Faulty Process", *Research Report*, No. 245, Yale University, Department of Computer Science, September 1982

[Han73] P. Brinch Hansen, "Operating System Principles", *Prentice Hall*, Englewood Cliffs, New Jersey, 1973

[Hec76] H.Hecht, "Fault-Tolerant Software for Real-Time Applications", *Computing Surveys*, Dezember 1976, S. 391-407

[Hoa74] C.A.R. Hoare, "Monitors: An Operating System Structuring Concept", *CACM*, vol. 17, No. 10, Oktober 1974, S. 549-557

[Hoa81] C.A.R. Hoare et al., "A Theory of Communicating Sequential Processes", *Technical Monograph PRG-16*, Oxford University Computing Laboratory, Programming Research Group, 1981

[Hof84] F. Hofmann, "Betriebssysteme: Grundkonzepte und Modellvorstellungen", erschienen in der Reihe *Leitfäden der angewandten Informatik*, Teubner-Verlag, Stuttgart, 1984

[Hor74] J.J. Horning et al., "A Program Structure for Error Detection and Recovery", *Proc. of an Inter. Symposium* held at Rocquencourt, Lecture Notes in

Computer Science No. 16, April 1974, S. 171-187 oder auch: *Proc. Conf. on Operating Systems*, IRIA, 1974, S. 177-193

[HR73] J.J. Horning, B. Randell, "Process Structuring", *Computing Susrveys*, vol. 5, No. 1, März 1973, S. 5-30

[JKR76] M.J. Jenson, K.H. Kim, D.L. Russell, "Language Tools for Fault Tolerant Programming", *Technical Memorandum PETP-1*, Program Error Tolerance Project, Electronic Sciences Laboratory, University of Southern California, Los Angeles, CA, November 1976

[Kan81] K. Kant, "Error Recovery in Concurrent Processes", *Dissertation*, The University of Texas at Dallas, Mai 1981

[Kim78a] K.H. Kim, "An Approach to Programmer-Transparent Coordination of Recovering Parallel Processes and its Efficient Implementation Rules", in *Proc. 1978 Int. Conf. on Parallel Processing*, August 1978, S. 58-68

[Kim78b] K.H. Kim, "Programmer-Transparent Coordination of Recovering Parallel Processes", *Technical Memorandum PETP-5*, Program Error Tolerance Project, Electronic Sciences Laboratory, University of Southern California, Los Angeles, CA, März 1978

[Kim78c] K.H. Kim, "Methodologies and Tools for Developing Robust FTSC Software", *Technical Report*, Departments of Electrical Engineering and Computer Science, University of Southern California, Los Angeles, CA, August 1978

[Kim78d] K.H. Kim et al., "Strategies for Structured and Fault-Tolerant Design of Recovery Programs", in *Proc. COMPSAC*, 1978, S. 651-656

[Kim79] K.H. Kim, "Error Detection, Reconfiguration and Recovery in Distributed Processing Systems", in *Proc. 1'st Int. Conf. on Distributed Computing Systems*, Oktober 1979, S. 284-295

[Kim80] K.H. Kim, "An Implementation of a Programmer-Transparent Scheme for Coordinating Concurrent Processes in Recovery", in *Proc. COMPSAC*, 1980, S. 615-621

[Kim82] K.H. Kim, "Approaches to Mechanization of the Conversation Scheme Based on Monitors", *IEEE Transactions on Software Engineering*, vol. SE-8, No. 3, Mai 1982, S. 189-197

[KS80] K.Kant, A. Silberschatz, "Error Recovery in Concurrent Processes", *Proc. of COMPSAC*, 1980, S. 608-614

[KS85] K. Kant, A. Silberschatz, "Error Propagation and Recovery in Concurrent Environments", *The Computer Journal*, vol. 28, No. 5, 1985, S. 466-473

[KSL84] C.M. Krishna, K.G. Shin, Y.H. Lee, "Optimization Criteria for Checkpoint Placement", *CACM*, vol. 27, No. 10, Oktober 1984, S.1008-1012

[Lap85] J.C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology", *Mitteilungen der Fachgruppe Fehlertolerierende Rechensysteme*, No. 4, Januar 1985, S. 16-35

[Lip74] R.J. Lipton et al., "A Comparative Study of Models of Parallel Computation", *15th Annual Symposium on Switching and Automata Theory*, New Orleans, Oktober 1974, S. 145-155

[Lom77] D.B. Lomet, "Process Structuring, Synchronization and Recovery using Atomic Actions", *Proc. ACM Conf. on Language Design for Reliable Software* (ebenso auch: *SIGPLAN Notices*, vol.12, No. 3), März 1977, S. 128-137

[LS81] B.W. Lampson, H. Sturgis, "Atomic Transactions", in Lecture Notes in Computer Science No. 105, Springer Verlag 1981, S. 246-265

[LS83] Y. Lee, K. Shin, "Analysis of Backward error Recovery for Concurrent Processes with Recovery Blocks", *Technical Report*, Computing Research Laboratory, Department of Electrical and Computer Engineering, The University of Michigan, Ann Arbor, Michigan, 1983

[Lyn81] N.A. Lynch, "Multilevel Atomicity - A New Correctness Criterion for Distributed Databases", *Technical Report*, School of Information and Computer Science, Georgia Institute of Technology, Mai 1981

[Mil85] A. Mili, "Towards a Theory of Forward Error Recovery", *IEEE Transactions on Software Engineering*, vol. SE-11, No. 8, August 1985, S. 735-748

[MK81] J.A. McPherson, C.R. Kime, "A Model for Fault Tolerant Process Maintenance", *11'th IEEE Symposium on Fault Tolerant Computing*, Portland, Maine, 1981, S. 109-111

[Mos81] J.E.B. Moss, "Nested Transactions: An Approach to Reliable Distributed Computing", *Dissertation*, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 1981

[Pap79] C.H. Papadimitriou, "The Serializability of Concurrent Database Updates", *Journal of the ACM*, vol. 26, No. 4, Oktober 1979, S. 631-653

[Ran75] B. Randell, "System Structure for Fault Tolerance", *IEEE Transactions on Software Engineering*, vol. 1, No. 2, 1975, S. 220-232

[Ran77] B. Randell, "System Structure for Software Fault Tolerance", in *Current Trends in Programming Methodology*, vol. 1, Herausgeber R.T. Yeh, Prentice Hall, Englewood Cliffs, NJ, 1977, S. 195-219

[RC85] D.J. Richardson, "Partition Analysis: A Method Combining Testing and Verification", *IEEE Transactions on Software Engineering*, vol. SE-11, No. 12, Dezember 1985, S. 1477-1490

[Ree78] D.P. Reed, "Naming and Synchronization in a Decentralized Computer System", *Ph. D. Thesis*, M.I.T. Department of Electrical Engineering and

Computer Science, September 1978

[Reu81] A. Reuter, "Fehlerbehandlung in Datenbanksystemen", Carl Hanser Verlag 1981

[RT79] D.L. Russell, M.J. Tiedemann, "Multiprocess Recovery Using Conversations", in *Dig. 9'th Int. Conf. on Fault Tolerant Comp.*, Madison, WI, Juni 1979, S. 106-109

[Rus76] D.L. Russell, "State Restoration among Communicating Processes", *Technical Report*, No. 112, Digital Systems Laboratory, Stanford University, 1976

[Rus77] D.L. Russell, "Process Backup in Producer-Consumer Systems", in *Proc. 6'th Symp. Operat. Syst. Principles*, West Lafayette, IN, November 1977; oder auch: *Operat. Syst. Rev.*, vol. 11, No. 5, S. 151-157

[Rus80] D.L. Russell, "State Restoration in Systems of Communicating Processes", *IEEE Transactions on Software Engineering*, vol. SE-6, No. 2, Maerz 1980, S. 183-194

[SB78] S.K. Shrivastava, J. Banatre, "Reliable Resource Allocation Between Unreliable Processes", *IEEE Transactions on Software Engineering*, vol. SE-4, No. 3, Mai 1978, S.230-241

[Sch85] R.D. Schlichting, "A Technique for Estimating Performance of Fault Tolerant Programs", *IEEE Transactions on Software Engineering*, vol. SE-11, No. 6, Juni 1985, S. 555-563

[Shr79a] S.K. Shrivastava, "Concurrent Pascal with Backward Error Recovery: Language Features and Examples", *Software Practice and Experience*, vol. 9, No. 12, Dezember 1979, S. 1001-1020

[Shr79b] S.K. Shrivastava, "Concurrent Pascal with Backward Error Recovery: Implementation", *Software Practice and Experience*, vol. 9, No. 12, Dezember 1979, S. 1021-1033

[SL84] K.G. Shin, Y.H. Lee, "Evaluation of Error Recovery Blocks Used for Cooperating Processes", *IEEE Transactions on Software Engineering*, vol. SE-10, No. 6, November 1984, S. 692-700

[SMI80] H. Sturgis, J. Mitchell, J. Israel, "Issues in the Design and Use of a Distributed File System", *Operat. Syst. Rev.*, vol. 14, No. 3, 1980

[Web83] K. Weber, "Modellierung von Fehlverhalten mit Berücksichtigung paralleler Abläufe", *Arbeitsbereichte des Instituts für Mathematische Maschinen und Datenverarbeitung*, Universität Erlangen-Nürnberg, Band 16, Nr. 9, 1983

[Wed83] H. Wedekind, "Datenbanksysteme I: Eine konstruktive Einführung in die Datenverarbeitung in Wirtschaft und Verwaltung", erschienen in der Reihe *Informatik*, Band 16, BI-Wissenschaftsverlag, Zürich 1983

[WH76] H. Wedekind, T. Härder, "Datenbanksysteme II", erschienen in der Reihe *Informatik*, Band 18, BI-Wissenschaftsverlag, Zürich 1976

[You74] J.W. Young, "A First Order Approximation to the Optimum Checkpoint Interval", *CACM*, vol. 17, No. 9, September 1974, S.530-531

Definierte Begriffe

Eine alphabetisch sortierte Liste der in dieser Ausarbeitung definierten Begriffe wird im folgenden angegeben:

Begriff	Seite
A	
Abbildung α	22
Ablauf	24
-> aktiver Ablauf	27
adr	23
Adreßraum eines Prozesses	23
Aktion α	22
-> Akzeptanztest ausführende Aktion	111
-> akzeptierte/verifizierte Aktion	(143) 215
-> fehlerhafte Aktion	140
-> globale Aktion	36
-> lesende Aktion	32
-> lokale Aktion	36
-> nachrichtenübertragende Aktion	38
-> nicht akzeptierte/verifizierte Aktion	215
-> schreibende Aktion	32
-> sichernde Aktion	70
-> synchronisierende Aktion	42
-> ungesicherte Aktion	140
-> zeitlich unteilbare Aktion	22
aktiver Ablauf	27

aktive Berechnung 26
aktuellster Wiederaufsetzpunkt 137
Akzeptanztest 111
Akzeptanztest ausführende Aktion 111
akzeptierte/verifizierte Aktion (143) 215
Anfangszustand 23
Atomarität 134
Ausführungsbedingung 23
Ausgabestelle einer Transition 99
Ausgabetransition 99

B

basiert auf Relation 47
Bedingungsteil 23
Bedingungsvariable 43
Befehlszähler 23
Befehlszähler eines Prozesses 23
bef_P 25
Berechnung 24
-> aktive Berechnung 26
bz 23

D

Dominoeffekt 177
dominoeffektfreies Prozeßsystem 176

E

Endezeitpunkt einer Aktion 135

Eingabestelle einer Transition 99

Eingabetransition 99

Einheitsrelation 138

F

Fehler 85
-> Rechensystemfehler 85
-> Prozeßsystemfehler 85

Fehlerbehandlungsfunktion 155

Fehlerbehandlungs-Modell 57

fehlerfreier Zustand 153

fehlerhafte Aktion 140

fehlerhafter Zustand (86) 151

Fehlersituation 1 87

Fehlersituation 2 87

Fehlerursacherermittlungs- und beseitigungsprozedur 115

Fehlerverursacher 141
-> potentielle Fehlerverursacher 142
-> nichtverifizierte potentielle Fehlerverursacher (216) 143
-> verifizierte potentielle Fehlerverursacher 143

G

global abhängig 49

globale Aktion 36

globale Variable 35

global lesende Aktion 34

global schreibende Aktion 34

H

Historie globaler Variablenwerte 71

Historie prozeßlokaler Wiederaufsetzpunkte 69

K

kommunikationsbedingt abhängig 50

kommunizierende Prozesse 54

Konfliktstelle 99

konsistenter Zustand (86) 151

kooperierende Prozesse 53

koordinierende Prozesse 55

koordinierungsbedingt abhängig 51

L

lauffähige Aktionen 25

lesende Aktion 32

lf_P 25

lokal abhängig 49

lokale Aktion 36

lokale Variable 35

M

markiertes Petri-Netz 100

Mehrprozessorsystem 62

Mehrrechnersystem 59

Monoprozessorsystem 57

N

Nachfolgermenge (einer Transition, Stelle) 99

Nachrichtenpuffer 37

Nachrichtensystem 37

nachrichtenübertragende Aktionen 38

Nachrichtenvariable 38

nichtverifizierte potentielle Fehlerverursacher (216) 143

P

parallele Wertzuweisung 22

Petri-Netz 99

$proc_P$ 25

potentielle Fehlerverursacher 141

Prozesse 25

-> unabhängige Prozesse 52

-> kooperierende Prozesse 53

-> kommunizierende Prozesse 54

-> koordinierende Prozesse 55

Prozeßablauf (Ablauf) 24

Prozeßsystem 23

Prozeßsystemfehler 85

Prozeßsystemklassen 52
Prozeßsystem-Modell 21
Prozeßsystem-Zustand 22

R

Rechensystem 57
-> Mehrprozessorsystem 62
-> Mehrrechnersystem 59
-> Monoprozessorsystem 57
-> verteiltes Mehrrechnersystem 59
-> zentralisiertes Mehrrechnersystem 59
Rechensystemfehler 85
Rechensystemverhalten 66
Recoveryeinheit 138
Recoveryfunktion 155
Recoveryverfahren 169
-> Rückwärtsrecoveryverfahren 169
-> Vorwärtsrecoveryverfahren 169
Rücksetzung 87

S

schreibende Aktion 32
Semaphorvariable 41
sichernde Aktion 70
Synchronisationssystem 42
Synchronisationsvariable 43
synchronisierende Aktion 42

T

Typisches Element 22

U

unabhängige Prozesse 52

unbeschränkbare Weitergabe nichtverifizierter Variablenwerte 205

unbeschränkbare Zahl nichtverifizierter Abhängigkeitsbeziehungen 205

unbeschränkbare zyklische Zurücksetzung 183

unbeschränkbar große Recoveryeinheiten 183

unbeschränkbar lange Prozeßinaktivphasen 183

unbeschränkbar lange Verifikationslücken 205

ungesicherte Aktion 140

ungesicherter Zustand 152

unteilbare Aktion 22

unvermeidbare Rücksetzung 181

V

val_P 24

Variable 23

-> akzeptierte/verifizierte Variable 215

-> Bedingungsvariable 43

-> globale Variable 35

-> lokale Variable 35

-> nachrichtenübertragende Variable 38

-> nicht akzeptierte/verifizierte Variable 215

-> Semaphorvariable 41

-> Synchronisationsvariable 43
Variable mit lesendem Zugriff 32
Variable mit schreibendem Zugriff 32
Verhalten 66
-> erwartetes Verhalten 66
-> fehlerhaftes Verhalten 66
-> nicht tolerierbares Verhalten 66
-> tolerierbares Verhalten 66
-> unerwartetes Verhalten 66
verifizierte potentielle Fehlerverursacher 143
verifizierte Aktion 143
vermeidbare Rücksetzung 181
verteiltes Mehrrechnersystem 59
Vorgängermenge (einer Transition, Stelle) 99
Vorwärtsrecoveryverfahren 169

W

Wertzuweisung 22
-> parallele Wertzuweisung 22
Wertzuweisungssymbol 22
Wiederanlaufverfahren 87
Wiederaufsetzpunkt 70
-> aktuellster Wiederaufsetzpunkt 137
-> betriebssysteminitiierter Wiederaufsetzpunkt 70
-> prozeßsysteminitiierter Wiederaufsetzpunkt 70

Z

zentralisiertes Mehrrechnersystem 59

zurückzusetzende Aktionen 144
Zustand 23
-> Anfangszustand 23
-> fehlerfreier Zustand 153
-> fehlerhafter Zustand (86) 151
-> konsistenter Zustand (86) 151
-> lokaler Zustand 69
-> prozeßlokaler Zustand 69
-> Prozeßsystem-Zustand 23
-> ungesicherter Zustand 152
-> verifizierter Zustand 152
Zustandsmenge 23
Zustandssicherungskonzept 69

Verwendete Symbole

Die in der vorliegenden Dissertation verwendeten Symbole sind in der folgenden Liste zusammenfassend dargestellt:

Symbol	Bedeutung
ϕ	leere Menge
¬	Negation
&	logische Konjunktion
||	logische Disjunktion
=>	logische Implikation, *wenn dann*
<=>	logische Äquivalenz, *genau dann wenn*
≡	Identität
λ	leere Folge
A^*	Menge der Folgen über A
A^+	Menge der Folgen über A ohne λ
$\mathbf{N}$	Menge der natürlichen Zahlen ohne {0}
$\mathbf{N}_0$	Menge der natürlichen Zahlen incl. {0}
$\mathbf{Z}$	Menge der ganzen Zahlen
$\cup$	mengentheoretische Vereinigung
$\cap$	mengentheoretischer Durchschnitt
-	mengentheoretische Differenz
$\bigcup_{x \in \{x_1, \ldots, x_n\}} M_x$	$M_{x_1} \cup M_{x_2} \cup \ldots \cup M_{x_n}$
$\bigcup_{i=1}^{n} M_i$	$M_1 \cup M_2 \cup \ldots \cup M_n$
$M_i \times M_j$	kartesisches Produkt der Mengen M_i und M_j

$\|M\|$	Kardinalität der Menge M
$\sum$	Summationssymbol
$\sum_{i=1}^{n} m_i$	$m_1+m_2+\ldots+m_n$
$\prod_{i=1}^{n} m_i$	$m_1 * m_2 * \ldots * m_n$
$\in$	Element von
$\notin$	nicht Element von
$=$	gleich
$<$	kleiner
$>$	größer
$\leq$	kleiner oder gleich
$\geq$	größer oder gleich
$\neq$	ungleich
$\subset$	echte Teilmenge von
$\supset$	echte Obermenge von
$\subseteq$	Teilmenge von
$\supseteq$	Obermenge von
$x \rightarrow y$	Abbildung von x nach y

Eingeführte Abkürzungen

Die im Rahmen dieser Arbeit eingeführten Abkürzungen sind in der folgenden Liste zusammenfassend in der Reihenfolge ihrer Einführung dargestellt (mit Abs und Def wird auf den entsprechenden Abschnitt bzw. die entsprechende Definition der Abkürzungseinführung verwiesen):

Abkürzung	Bedeutung
$WP_{i,j}$	j-ter Wiederaufsetzpunkt des Prozesses P_i (Abs. 1.3)
$AT_{i,j}$	j-ter Akzeptanztest des Prozesses P_i (Abs. 1.3)
::=	Einführung neuer Begriffe oder Funktionen durch schon vorhandene, *definiendum*::=*definiens* (Abs. 2.1.1.1)
::<=>	Einführung neuer Prädikate oder Aussagen als Abkürzungen für schon vorhandene, *definiendum*::<=>*definiens* (Abs. 2.1.1.1)
:=	Wertzuweisungssymbol (Abs. 2.1.1.1)
D	Zustandsmenge (Def. 2.1.1.2/1)
w	Anfangszustand, $w \in D$ (Def. 2.1.1.2/1)
A	endliche Menge von Aktionen (Def. 2.1.1.2/1)
$P=(A,D,w)$	Prozeßsystem (Def. 2.1.1.2/1)
bz(...)	Funktion, die jede Aktion genau einem Prozeß zuordnet (Def. 2.1.1.2/1)
adr(...)	Funktion, die jeder Aktion genau eine Aktionsadresse zuordnet (Def. 2.1.1.2/1)

B_α	Bedingungsteil der Aktion α (Def. 2.1.1.2/1)
$N_\alpha(\ )$	Funktion, die eine Nachfolgeraktion zur Aktion α berechnet (Def. 2.1.1.2/1)
$F_\alpha(\ )$	Funktion, die Berechnungen auf Variablen der Aktion α durchführt und ihre Werte ändert, ausgenommen des α zugeordneten Befehlszählers (Def. 2.1.1.2/1)
$bz_{bz(\alpha)}$	zur Aktion α zugeordneter Befehlszähler (Def. 2.1.1.2/1)
D_i	Adreβraum des Prozesses P_i (Def. 2.1.1.2/1)
bz_i	Befehlszähler des Prozesses P_i (Def. 2.1.1.2/1)
D_{V_r}	Wertebereiche sämtlicher Variablen eines Prozeβsystems, Befehlszähler ausgenommen (Def 2.1.1.2/1)
V_r	Namen sämtlicher Variablen eines Prozeβsystems (Def. 2.1.1.2/1) ohne die Befehlszähler
α	endliche Aktionenfolge (Berechnung, Ablauf); analog werden die griechischen Kleinbuchstaben β und γ verwendet (Abs. 2.1.1.3)
λ	leere Aktionenfolge (Berechnungsfolge) (Abs. 2.1.1.3)
$\alpha[i]$	i-tes Element der Aktionenfolge α (Abs. 2.1.1.3)
α_i	Anfangsstück von α bis einschließlich dem i-ten Element $\alpha[i]$ (Abs. 2.1.1.3)
$\alpha\beta$	Konkatenation der Abläufe α und β (Abs. 2.1.1.3)
$\|\alpha\|$	Länge der Folge α (Abs. 2.1.1.3)
$W(\alpha)$	$W(\alpha)::=\{\alpha[1],\ldots,\alpha[\|\alpha\|]\}$ (Abs. 2.1.1.3)

$val_P(\ldots)$	Funktion, welche Ablaufen des Prozeßsystems P Zustände zuordnet (Def. 2.1.1.3/2)
$proc_P(\ldots)$	Äquivalenzrelation auf der Aktionenmenge A eines Prozeßsystems P (Def. 2.1.1.3/3)
P_i	Prozeß P_i, i-te Äquivalenzklasse von $proc_P$ (Abs. 2.1.1.3)
$bef_s(\ldots)$	Menge derjenigen Aktionen, deren Adressen mit den aktuellen Befehlszählerständen übereinstimmen (Def. 2.1.1.3/3)
$lf_s(\ldots)$	Menge derjenigen Aktionen, die lauffähig sind, d.h. deren Ausführungsbedingung erfüllt ist (Def. 2.1.1.3/3)
$L(\ldots)$	Prädikat "ist Leseaktion" (Def. 2.1.2.1/1)
$S(\ldots)$	Prädikat "ist Schreibaktion" (Def. 2.1.2.1/1)
LV_x	Variablen, auf die x lesend zugreift (Def. 2.1.2.1/2)
SV_x	Variablen, auf die x schreibend zugreift (Def. 2.1.2.1/2)
$gL(\ldots)$	Prädikat "ist globale Leseaktion" (Def. 2.1.2.1/2)
$gS(\ldots)$	Prädikat "ist globale Schreibaktion" (Def. 2.1.2.1/3)
V_x	Variablen, auf die x zugreift (Def. 2.1.2.2/1)
$l(\ldots)$	Prädikat "ist nur lokal" (Def. 2.1.2.2/2 und 3)
$g(\ldots)$	Prädikat "ist nur global" (Def. 2.1.2.2/2 und 3)
lV_x	Variablen, auf die x zugreift und die nur lokal definiert sind (Def. 2.1.2.2/2)
gV_x	Variablen, auf die x zugreift und die nur global definiert sind (Def. 2.1.2.2/2)

lv_i	lokale Variable (Abs. 2.1.2.2)
gv_i	globale Variable (Abs. 2.1.2.2)
lA_x	Aktionen von x, die nur auf lokale Variablen zugreifen (Def. 2.1.2.2/3)
gA_x	Aktionen von x, die (auch) auf globale Variablen zugreifen (Def. 2.1.2.2/3)
np_i	Nachrichtenpuffer (Def. 2.1.2.3/1)
nv_i	Variable vom Typ Nachricht (Def. 2.1.2.3/1)
D_N	Definitionsbereich vom Typ Nachricht (Def. 2.1.2.3/1)
$n(\ )$	Prädikat "ist nachrichtenübertragend" (Def. 2.1.2.3/1 und 2) bzw. "kommunikationsbedingt abhängig" (Def. 2.1.3.3/1)
$s(\ )$	Prädikat "ist synchronisierend" (Def. 2.1.2.4/1 und 2) bzw. "koordinierungsbedingt abhängig" (Def. 2.1.3.4/1)
$b(\ )$	Prädikat "ist Bedingungsvariable" (Def. 2.1.2.4/3)
sv_i	Semaphorvariable (Def. 2.1.2.4/1)
$\Phi_\alpha(\ .\ )$	auf Semaphorvariablen definierte Ausführungsbedingung der Aktion α (Def. 2.1.2.4/1)
$\Theta_\alpha(\ )$	Menge der auf Semaphorvariablen definierten Zuweisungen der Aktion α (Def. 2.1.2.4/1)
B_α	*basiert auf* Relation, definiert auf Aktionen der Folge α, $B_\alpha \subseteq W(\alpha) \times W(\alpha)$ (Def. 2.1.3.1/1)
$l(P)$	Kennzeichnung von P als System voneinander unabhängiger Prozesse, d.h. Prozesse mit ausschließlich lokalen Beziehungen (Def. 2.1.4.1/1)

$g(P)$	Kennzeichnung von P als einem System miteinander (ohne Einschränkungen) kooperierender Prozesse (Def. 2.1.4.2/1)
$CL(P)$	Menge aller Prozeßsysteme (Abs. 2.1.4.2)
$CL_x(P)$	Menge aller Prozeßsysteme des Typs x (Def. 2.1.4.1/1, 2.1.4.2/1, 2.1.4.3/1, 2.1.4.4/1)
$n(P)$	Kennzeichnung von P als einem System kommunizierender Prozesse (Def. 2.1.4.3/1)
$s(P)$	Kennzeichnung von P als einem System koordinierender Prozesse (Def. 2.1.4.4/1)
$val_P(*)$	der aktuelle Systemzustand von P (Def. 2.2.2.1/1)
$val_P(*\|P_i)$	der aktuelle lokaler Systemzustand von P eingeschränkt auf den Prozeß P_i (Def. 2.2.2.1/1)
$h(P_i)$	Liste aller gesicherten lokalen Zustände des Prozesses P_i (Abs. 2.2.2.1)
$up(P_i)$	elementare Aktion, die eine Erweiterung von $h(P_i)$ um den aktuellen lokalen Prozeßzustand $val_P(*\|P_i)$ bewirkt (Def. 2.2.2.1/2)
$up(\ldots)$	Prädikat "ist sichernd" (Def. 2.2.2.1/3)
$h(gv_i)$	zur globalen Variablen gv_i geführte Liste aller Wertänderungen (Abs. 2.2.2.2)
$P=(S,T,K)$	Petri-Netz (Def. 2.2.3.4/1)
(P,M)	markiertes Petri-Netz (Def. 2.2.3.4/3)
$at(\ldots)$	Prädikat "ist akzeptanztestausführende Aktion" (Def. 2.2.4.1/1)
$t(\ldots)$	Funktion zur Ermittlung des Endezeitpunktes einer Aktion (Def. 3.1.1/1)

$WP_\alpha(\ldots)$	Funktion zur Bestimmung des aktuellsten Wiederaufsetzpunktes zu einer Aktion (Def. 3.1.2/1)
E_α	*Einheitsrelation* auf Aktionen der Folge α, $E_\alpha \subseteq W(\alpha) \times W(\alpha)$ (Def. 3.1.4/1)
$f_\alpha(\ )$	Prädikat "ist fehlerhaft" (Def. 3.2.1/1, 3.3.2/1)
$u_\alpha(\ )$	Prädikat "ist ungesichert" (Def. 3.2.1/2, 3.3.2/2)
$R_1 \circ R_2$	Komposition der Relationen R_1 und R_2 (Def. 3.2.2/1)
$R_1 \cup R_2$	Vereinigung der Relationen R_1 und R_2 (Def. 3.2.2/2)
R^*	transitive Hülle der Relation R (Def. 3.2.2/3)
$F_\alpha(\ldots)$	Menge potentieller Fehlerverursacher (Def. 3.2.2/4)
$v_\alpha(\ldots)$	Prädikat "ist verifiziert" (Def. 3.2.3/2, 3.3.3/1)
$V_v(\ldots)$ $(V_{nv}(\ldots))$ $LV_v(\ldots)$ $(LV_{nv}(\ldots))$ $SV_v(\ldots)$ $(SV_{nv}(\ldots))$	verifizierte (nichtverifizierte) Variablen, auf die lesend bzw. schreibend zugegriffen wird (Def. 3.2.3/1)
$F_{nv(\alpha)}(\ldots)$	Menge der nichtverifizierten potentiellen Fehlerverursacher (Def. 3.2.3/3, 4.3.3/7)
$F_{v(\alpha)}(\ldots)$	Menge der verifizierten potentiellen Fehlerverursacher (Def. 3.2.3/3)
$z_\alpha(\ldots)$	Prädikat "ist zurückzusetzen" (Def. 3.2.4/1)
$Z_\alpha(\ldots)$	Menge von Aktionen der Folge α, die zurückgesetzt werden müssen (Def. 3.2.4/2)
$r(\ldots)$	Recoveryfunktion (Fehlerbehandlungsfunktion) r (Def. 3.4.1/1)
$r_i(\ldots)$	Recoveryfunktion (Fehlerbehandlungsfunktion) r_i (Abs. 3.4)

$U_{Z_\alpha}(\ )$	Menge der Aktionen, deren Rücksetzung unvermeidbar ist (Def. 4.2.1/1)
$V_{Z_\alpha}(..)$	Menge der Aktionen, deren Rücksetzung vermeidbar ist (Def. 4.2.1/1)
ZT_α	Aktion α zugeordnete Zuweisungsteilrelation (Def. 4.3.3/1)
BT_α	Menge der Variablen, die zum Bedingungsteil der Aktion α gehören (Def. 4.3.3/2)
$AT_{\alpha_{i,j}}(...)$	Prädikat "ist in der Lage zu verifizieren/validieren" (Def. 4.3.3/3)
$A_{\alpha,[v]}$	***akzeptiert*** Relation auf der Folge α bzgl. der Variablen v (Def. 4.3.3/4)
$N_{\alpha,[v]}$	***akzeptiert nicht*** Relation auf der Folge α bzgl. der Variablen v (Def. 4.3.3/5)
$B_{\alpha,[v]}$	***basiert auf*** Relation auf Variablenebene (Def 4.3.3/6)
g_{re}	maximale Größe einer Recoveryeinheit, die in aktiven Berechnungen eines gegebenen Prozeßsystems auftreten kann (Abs 5.1)
g_{re_α}	maximale in der aktiven Berechnung α vorkommende Recoveryeinheitsgröße (Abs. 5.1)

LEBENSLAUF

Peter Bathelt

12.9.1955	geboren in Nachod/CSSR, Vater Dipl. Ing. Walter Bathelt, Mutter Drahomira Bathelt
1962 - 68	Besuch der Volksschule in Male Svatonovice/CSSR (1962 - 66), Augsburg (1966 - 67), Nürnberg (1967 - 68).
1968 - 75	Besuch des Johannes Scharrer Gymnasiums in Nürnberg, eines wirtschaftswissenschaftlichen Gymnasiums mit 7-jähriger Schulform.
1975	Abitur
1975 - 76	Bundeswehr
1976 - 81	Studium der Informatik an der Universität Erlangen-Nürnberg mit Nebenfach Betriebswirtschaft. Schwerpunkt Betriebsprogrammierung; weitere Prüfungsfächer sind Automatentheorie, Mustererkennung, Wahrscheinlichkeitsrechnung und Operations Research. Studienarbeit am Lehrstuhl von Professor Dr. H.J. Schneider: "Beseitigung der Programmrekursion nach der Fixpunktmethode". Diplomarbeit bei Professor Dr. F. Hofmann: "Vergleich von Synchronisationsmechanismen".
November 1981	Abschluß der Diplomprüfung mit Auszeichnung.
seit Januar 1982	Mitarbeiter bei der Siemens AG, Erlangen, in der systemtechnischen Entwicklung des Unternehmensbereichs Energie- und Automatisierungstechnik.
seit Januar 1983	Kontakte zum Institut für Mathematische Maschinen und Datenverarbeitung IV der Universität Erlangen-Nürnberg; Vorbereitung auf die Promotion.

Offsetdruck · Fotodruck · Hogl · Hauptstraße 109 · 8520 Erlangen · Tel.09131/22990